VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2018 Jiří Novák

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student:	**Jiří Novák**
Studijní program:	B2647 Informační a komunikační technologie
Studijní obor:	2612R025 Informatika a výpočetní technika
Téma:	Absolvování individuální odborné praxe Individual Professional Practice in the Company
Jazyk vypracování:	čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Tieto Czech s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Daniel Stříbný**

Konzultant bakalářské práce: Mgr. Katarína Farkasová

Datum zadání: 01.09.2017
Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 12. dubna 2018 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských/magisterských programech VŠB-TU Ostrava.

Tieto Czech s.r.o.
28. října 3346/91
702 00 Ostrava - Moravská Ostrava
IČO 64608051 DIČ CZ64608051

V Ostravě dne 11. dubna 2018 .....

Abstrakt

Práce pojednává o absolvování mé odborné praxe ve firmě Tieto Czech s.r.o..
Cílem této práce je ve stručnosti představit firmu, v níž má odborná praxe probíhala a pozici, kterou jsem vykonával. Dalším bodem je popsání zadaných úkolů a jejich řešení.
V poslední části jsou popsané uplatněné znalosti a celkové zhodnocení přínosů absolvování praxe.

Bakalářská práce popisuje mou pracovní pozici GUI automatizačního testera týmu NMA a úkoly, se kterými jsem se střetával, jako práce se SoapUI, migrace testů a jejich následnou opravu, tvorba nových testů pro kontextovou nabídku a instalace nového testovacího nástroje včetně konfigurace serveru.

Klíčová slova: testování, automatizace, GUI, Selenium, SoapUI

Abstract

This thesis describes my practice in Tieto Czech s.r.o. company. The aim of this thesis is to briefly introduce the company in which my professional practice took place and my position. The next point is describe of assigned tasks and their solutions. The last part describes applied knowledge and overall assessment of benefits gained by professional practice.

This bachelor thesis describes my position of GUI automation tester of the NMA team and tasks, which I encountered, such as working with SoapUI, migration of tests and their subsequent repairs, creating new tests for the context menu, and installing a new test tool including server configuration.

Key Words: testing, automation, GUI, Selenium, SoapUI

Obsah

Seznam použitých symbolů a zkratek

API	– Application Programming Interface
DB	– Databáze
DNS	– Domain Name System
GUI	– Graphical User Interface
HTML	– HyperText Markup Language
HTTP	– Hypertext Transfer Protocol
Hub	– Server řídící node servery
ID	– Identifikační číslo
IIS	– Internet Information Services
IT	– Informační technologie
JDBC	– Java Database Connectivity
JAR	– Java ARchive
JS	– JavaScript
JSON	– JavaScript Object Notation
MXD	– Map Exchange Document
NMA	– Network Management Automation
Node	– Server hostující webové prohlížeče
OS	– Operační systém
REST	– Representational state transfer
RVM	– Ruby Version manager
SOAP	– Simple Object Access Protocol
SQL	– Structured Query Language
SSH	– Secure Shell
SVN	– Apache Subversion
TC	– Test Case
TFS	– Team Foundation Server
TS	– Test Suit
XML	– Extensible Markup Language
XPath	– XML Path Language

Seznam ilustrací a seznam grafů

Seznam výpisů zdrojového kódu

Úvod

Ve firmě Tieto pracuji již od druhého ročníku bakalářského studia, tehdy jako Windows administrátor. Delší dobu jsem uvažoval o změně své pracovní pozice a možnosti vykonat praktickou bakalářskou práci. To tedy bylo důvodem, proč jsem přestoupil do nového týmu na úplně jinou pracovní pozici jako tester zaměřený na automatizované GUI testy.

V prvních kapitolách popisuji firmu, můj tým a pracovní pozici, kterou jsem vykonával.

V následujících kapitolách naleznete popis testování v NMA týmu, popsané technologie, které jsem při vývoji automatizovaných testů používal, svěřené úkoly, jejich řešení a problémy, které se při řešení objevovaly.

Konkrétně půjde o migraci testovacího prostředí na nový server, oprava automatizovaných testů, vytváření nových testů, práci na kontextové nabídce a migrace testovacího nástroje z fyzického serveru do virtuálního.

V závěrů práce ohodnotím můj přínos, znalosti, které jsem získal jak při studiu, tak při vykonávaní praxe, případně samostudiem a celkové zhodnocení dosažených výsledků.

1 O firmě

Tieto je největší severoevropská IT společnost poskytující komplexní IT servis. Zajišťuje také služby v oblasti vývoje produktů pro firmy působící v odvětví komunikací a integrovaných technologií. Na základě svých znalostí, technologických vizí a inovativního myšlení se společnost Tieto aktivně snaží inspirovat a zapojit své zákazníky do hledání nových způsobů, jak zefektivnit jejich podnikatelskou činnost. [1]

Do České republiky společnost Tieto vstoupila v roce 2001 a v roce 2004 otevřela své softwarové centrum v Ostravě. S více než 2 200 zaměstnanci je jedním z největších zaměstnavatelů v oblasti IT služeb v České republice a největším v Moravskoslezském kraji. Z hlediska počtu kmenových zaměstnanců je česká pobočka třetí největší pobočkou Tieto korporace na světě. První dvě místa zaujímají mateřské země Finsko a Švédsko. [1]

1.1 O NMA týmu

V dnešním světě, který je závislejší a závislejší na nejrůznější technologie, je čím dál tím více důležité mít dokonalý přehled a správu nad dodávkami energie, řešení energetických výpadků, výstavbu elektrické sítě a její údržba a také nacenění energie.

Všechny výše zmíněné problémy řeší Network Management Automation tým, který je zodpovědný za vývoj a testování softwaru určeného pro energetické společnosti sídlící ve Finsku, Švédsku a Norsku.

1.2 NMA nejdůležitější produkty:

- **Titanium** – hlavní produkt, mapovací software určený pro záznam, výstavbu a plánování údržby elektrické sítě, postavený na webových technologií a ArcGISMap.
- **PG Field** – desktopová aplikace, která je určena pro použití v terénu na tabletech. Spolupracuje s aplikací Titanium.
- **Project & Budgeting** – webová aplikace určená pro naceňování výstavby elektrické sítě, spolupracuje s aplikací Titanium.

1.3 Má pracovní pozice

Do týmu jsem nastoupil jako tester zaměřený na automatizované GUI testy, mojí pracovní náplní bylo vytváření nových automatizovaných testů, jejich údržba, starání se o testovací prostředí, v podstatě jsem byl jediná osoba, která byla za GUI testy zodpovědná.

Každé ráno bylo mou první povinností zhodnotit výsledky GUI testů, které jsem pak prezentoval na Scrum schůzce, což je metoda agilního vývoje softwaru uplatňovaná v NMA týmu. Zhodnocení spočívalo v tom, zda testy nalezly relevantní chyby případně jestli se nejednalo o falešný poplach, tedy test, který je třeba opravit.

Po scrum schůzce byl zbytek mého pracovního dne věnován na opravu vadných testů, vytváření nových či práce na migraci testovacího prostředí a testovacího nástroje.

1.4 Testování v NMA týmu

Testování v NMA týmu je rozdělené na manuální a automatizované, kde manuální testování provádí testy, které nelze zautomatizovat či s velkými obtížemi. Automatické testy se dělí na testovaní REST služeb a GUI testování, které testuje hlavní produkt Titanium a pak také Project & Budgeting.

2 Použité technologie

2.1 Selenium

Jedná se o otevřený multiplatformní framework, který umožňuje řídit chování webového prohlížeče a jeho akce. Existuje mnoho implementací, např. pro Java, .NET, Ruby, Python, JavaScript … [2]

Jelikož Selenium jako takové je implementováno ve formě http metod, lze snadno vytvořit framework pro nový programovací jazyk či integrovat Selenium do jiných produktů. [3]

2.2 Test Runner

Jde o software, který řídí vykonávání testů, jejich správu a uložení výsledků. V NMA týmu se používá vlastní napsaný nástroj, který je naprogramovaný v Ruby a je určen pro běh pod OS Linux.

Funguje na principu master a slave, test runner je master a ostatní slaves mu nabízejí službu, v tomto případě se jako služba myslí možnost spuštění webového prohlížeče.

2.3 SoapUI

Aplikace je určená pro vytváření servisových testů [4], lze ji ale také využít ve spolupráci se Selenium frameworkem, tím nám aplikace umožní vytvářet GUI testy pomocí jednoduchého skládání již hotových „modulů“, tímto se dramaticky snižuji nároky na testera, protože není třeba znát programovací jazyk pro vývoj GUI testů.

Přesto ale SoapUI umožnuje využít např. programovací jazyk Groovy [5], což zjednodušuje vývoj obzvláště složitých GUI testů.

2.4 XPath

Je dotazovací jazyk, pomocí kterého lze adresovat různé části XML dokumentu a vybírat z něj jednotlivé elementy a pracovat s jejich hodnotami a atributy. [6]

Používá se pří vývoji GUI testů, kdy je třeba interagovat s elementy HTML dokumentu, které nelze jednoduše adresovat. [7]

2.5 Groovy

Je objektově orientovaný skriptovací jazyk pro platformu Java. [8]
Je nabízen SoapUI jako možnost zjednodušení vývoje složitých GUI testů.

2.6 SQL

Je dotazovací jazyk, který je používán pro práci s daty v relačních databázích.
Používá se pří vývoji GUI testů, kdy je potřeba manipulovat s daty v DB.

3 Svěřené úkoly

3.1 Migrace testovacího prostředí na nový server

Hned při nástupu jsem byl pověřen migrací testovacího prostředí. Důvodem migrace byla skutečnost, že GUI testy testovaly Titanium spolu se servisovými testy, což přinášelo problémy, snaha byla tedy o získání nezávislosti GUI testů.

Vytvoření nového prostředí nebylo nijak složité jako spíše hodně pracné a náročné na správnou konfiguraci. Poté bylo třeba upravit testy tak, aby byly spuštěné vůči novému prostředí.

Časová náročnost: 7 dní

3.2 Oprava nefunkčních testů po migraci

Hned poté jsem mohl pracovat na opravu GUI testů, které drtivě spoléhaly na data, která byla dodána servisovými testy. Navíc došlo ke změně databáze a bylo třeba také upravit connection string. Oprava testů nebyla nijak jednoduchá, protože mnoho testů bylo napsáno špatně a bylo třeba je opravit.

Časová náročnost: 7 dní

3.3 Vytvoření testů testující kontextovou nabídku

Byl jsem požádán o vytvoření nové sady testů, které by kontrolovaly kontextovou nabídku aplikace Titanium. Vytvoření testů si však žádalo vytvoření pomocných modulů a poté vymyšlení způsobu, jak testy správně strukturovat. Testy jsou celkem složité a časově náročné na napsaní. Navíc jsem narazil na skutečnost, že testovací prostředí neobsahuje hromadu nutných dat potřebných k testům.

Časová náročnost: 14 dní

3.4 Migrace testovacího nástroje na virtuální server

Kromě migrace testovacího prostředí jsem podědil po svém předchůdci i plán migrace testovacího nástroje. Rozhodl jsem se, že migraci začnu a dokončím, i když se nejednalo o akutní problém.

Impulsem byl fakt, že mi selhal slave – node a já si uvědomil, že selhat může i master – hub, kde žádnou zálohu nemáme. Navíc byl testovací nástroj značně nestabilní a jelikož byl pro tuto migraci vyhrazen virtuální server, tak plán byl jasný. Nainstalovat a nakonfigurovat nový testovací nástroj, co nejlépe včetně konfigurace serveru.

Časová náročnost: 14 dní

4 Migrace testovacího prostředí na nový server

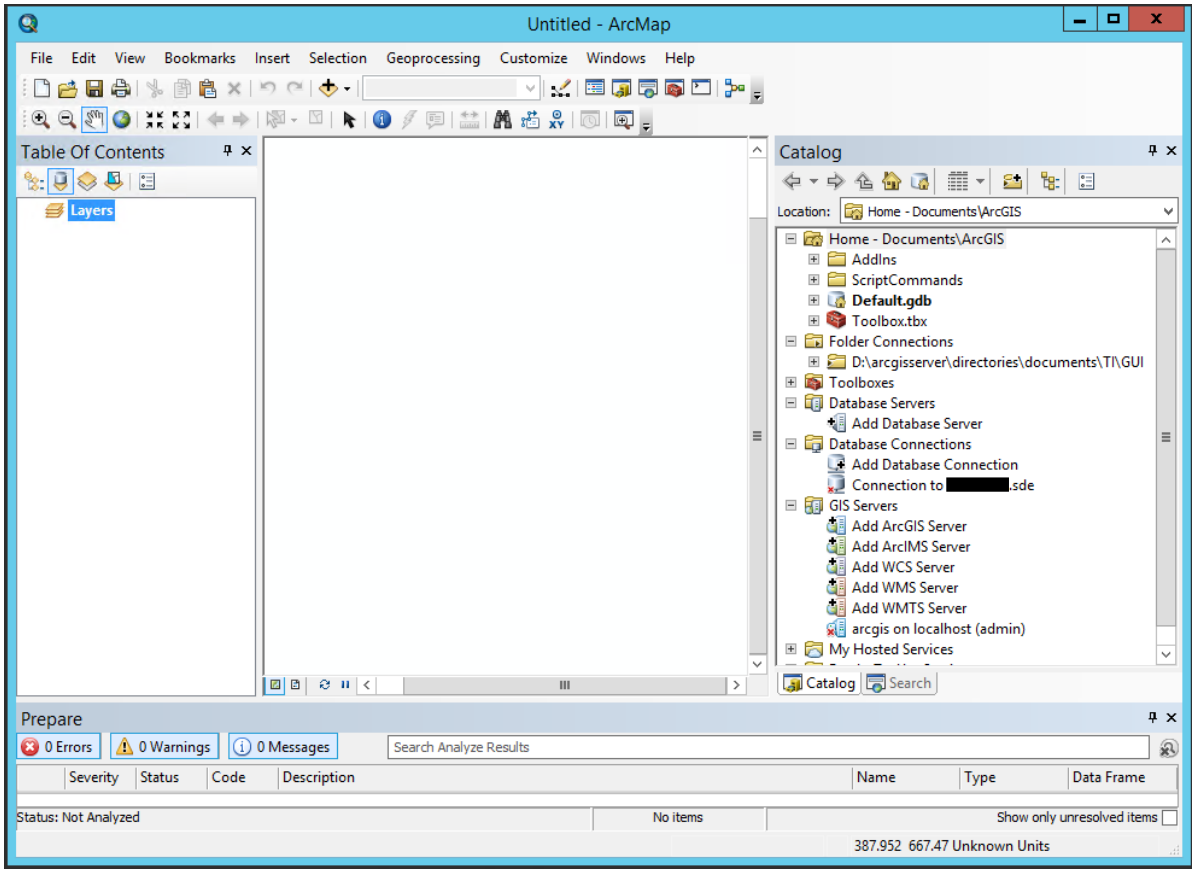
Jako první úkol, který jsem dostal byla migrace testovacího prostředí. Toto rozhodnutí učinil můj předchůdce na základě problémů, se kterými se setkával. Jeden z problémů byla závislost GUI testů na servisové testy, protože obě varianty automatických testů běžely vůči stejnému prostředí aplikace Titanium a stejné databáze. Servisové testy se spouštěly před GUI testy a mnoho testů bylo napsaných tak, že byly závislé na správné provedení servisových testů. Stávalo se tedy, že pokud servisové testy z nějakého důvodu neprošly, tak potom neprošly ani GUI testy.

Dalším problémem byl fakt, že když byla potřeba testy manuálně spustit, tak bylo třeba dávat pozor, zda už neběží servisové testy či naopak. Proto byla migrace schválena a jelikož můj předchůdce opustil pozici GUI testera, dostal jsem za úkol tedy migraci udělat.

Potíž byla také se samotným serverem, který nezvládal nápor GUI testů, kvůli toho Titanium reagovalo na testy velmi pomalu, čímž se způsobovala falešná selhávání testů.

4.1 Konfigurace prostředí – ArcGIS služby

První věcí, co bylo nutné udělat, bylo vypublikování ArcGIS služeb, které Titanium používá pro vykreslování mapy. Musel jsem se tedy přihlásit na server, na kterém toto vše poběží, a otevřít ArcMap.

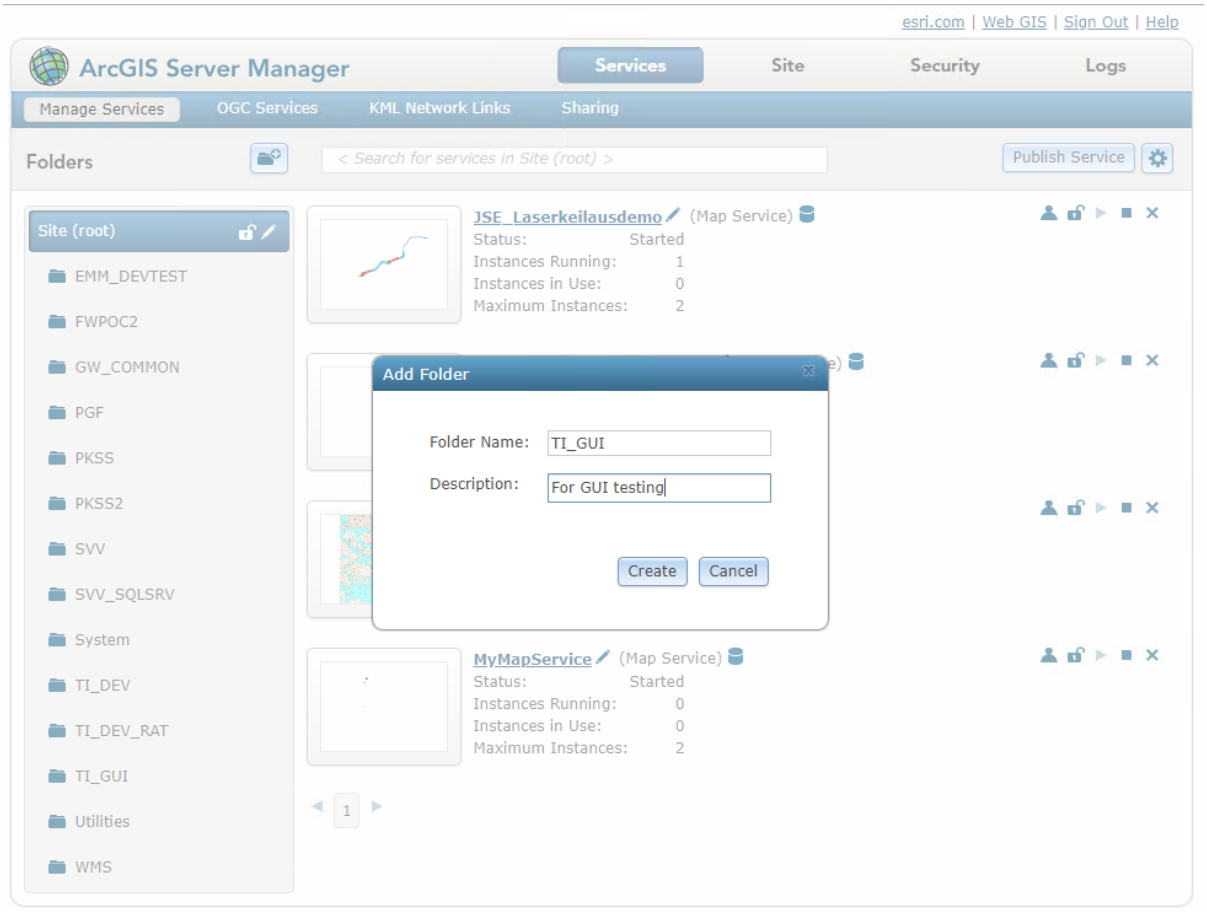


Obrázek 1: ArcMap

ArcMap je tedy nástroj od firmy Esri, který slouží k vytváření mapových podkladů a přes který je možné vytvářet mapové služby – ArcGIS služby [10], které poskytují aplikaci Titanium mapu.

Tyhle služby jsou velmi náročné na výkon serveru, a navíc vyžadují připojení do databáze. Proto bylo nutné znovu je publikovat, protože poběží na jiném serveru včetně nové databáze.

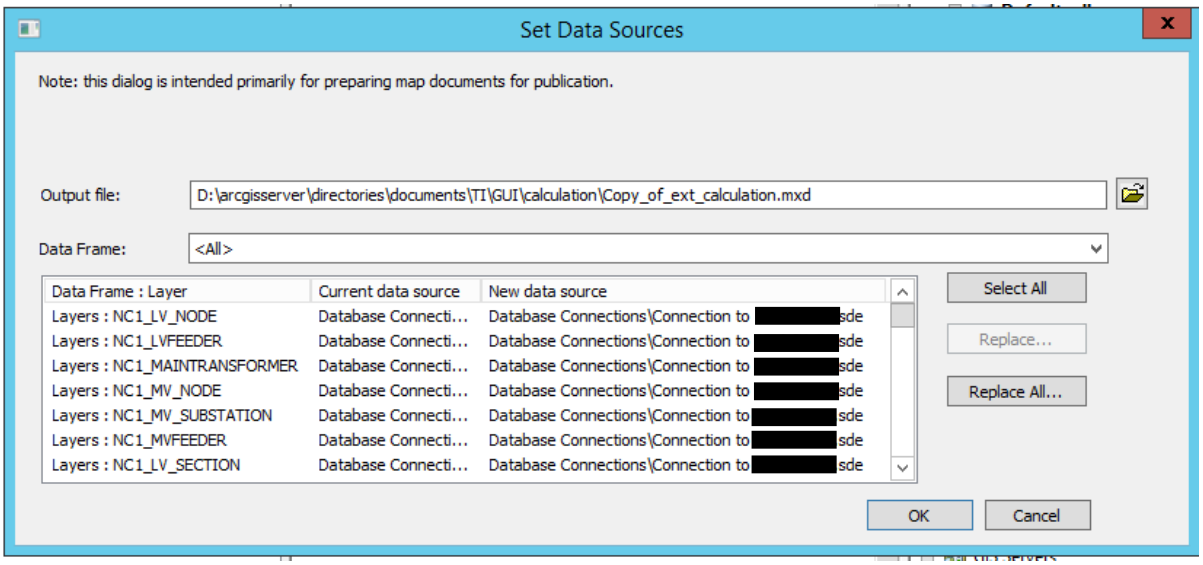
Musel jsem vytvořit složku, do které se budou publikovat služby. To se udělá tak, že přihlásím na ArcGIS Server Manager, což je webová aplikace umožňující jejich správu.



Obrázek 2: ArcGIS Server Manager

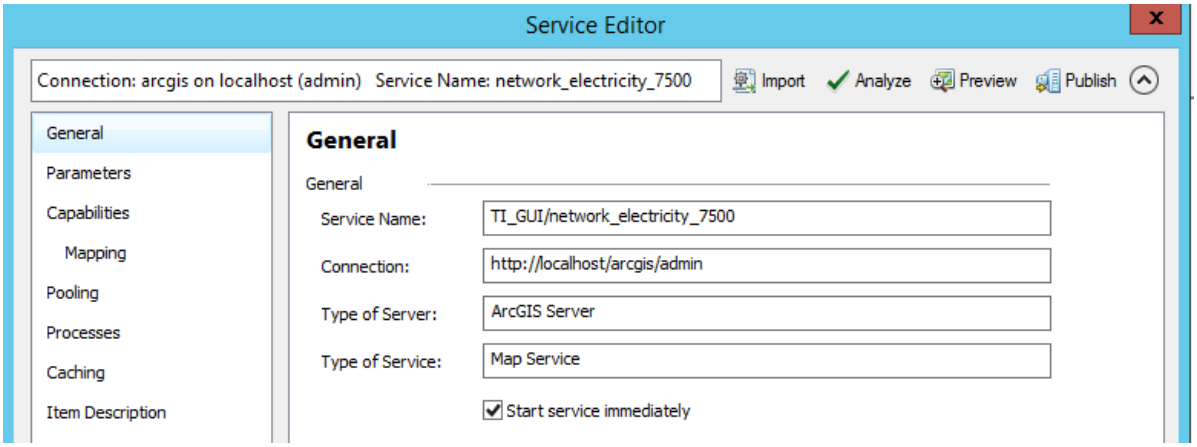
Teď je třeba nakopírovat mxd soubory, které poskytují vrstvy pro mapu a které jsou zdrojem informací pro vytvoření služeb [11]. Soubory jsem nakopíroval z vývojářského serveru a poté načetl do aplikace ArcMap.

Každy mxd soubor má nastavený datasource, který říká, kde má mxd soubor uložené mapové vrstvy. Jelikož jsem mxd soubory nakopíroval z vývojářského serveru, je nutné jim upravit datasource na připojení do správné databáze. Nastavení se provede jednoduše kliknutím na mxd soubor a výběr volby Set Data Source. Pak se musí ke každé vrstvě nastavit připojení do databáze.



Obrázek 3: ArcMap – nastavení data source

Po nastavení správného datasource, jsem mohl konečně přejít k samotnému publikování. Stačí na mxd soubour kliknout pravým a vybrat volbu Share As Service. Otevře se průvodce, ve kterém nastavím, na kterém ArcGIS serveru chci publikovat, do které složky a co všechno má ArcGIS služba poskytovat.



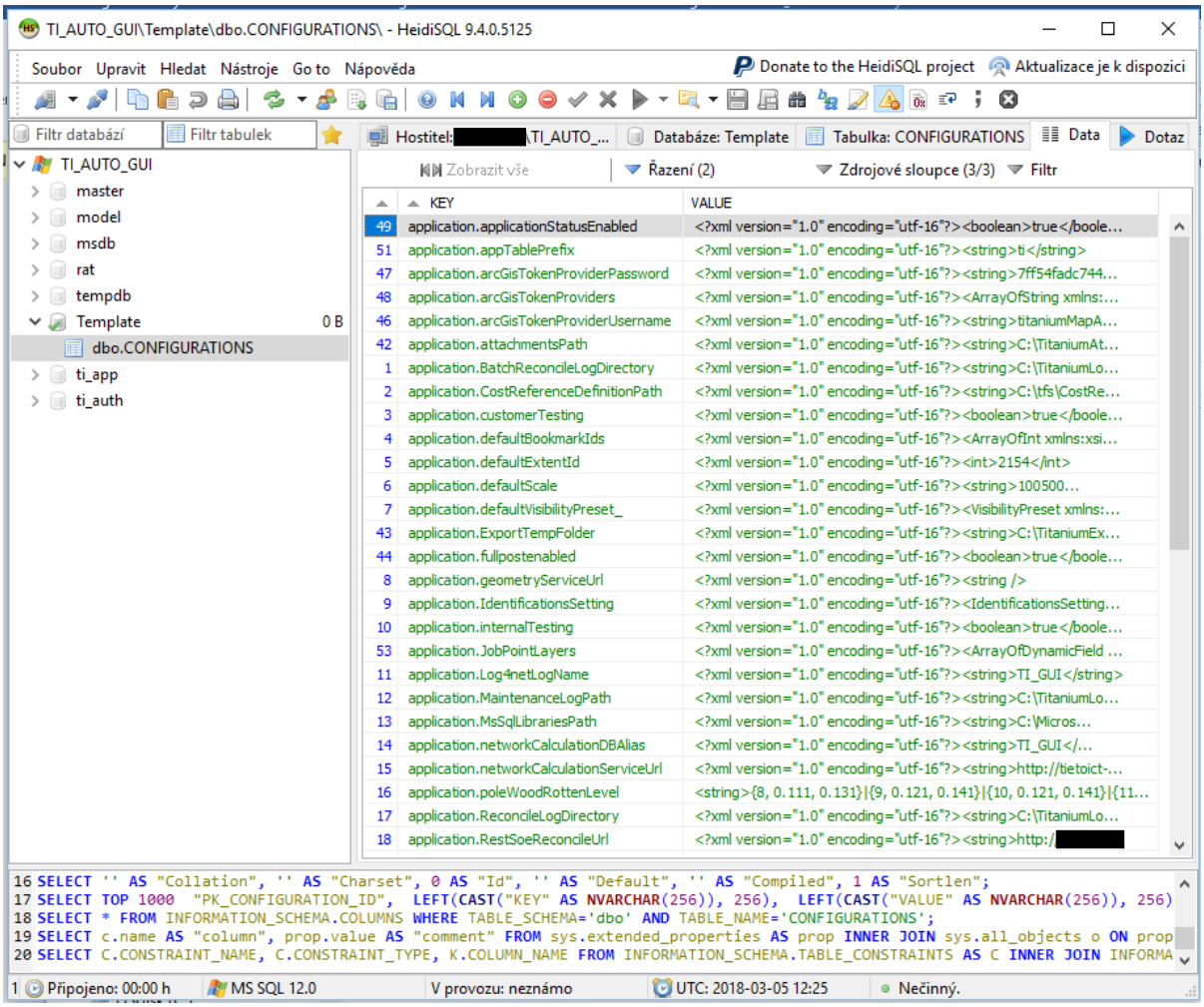
Obrázek 4: ArcMap – Service Editor

Ještě před kliknutím na tlačítko Publish bylo dobré prvně provést analýzu, která dokázala odhalit případné problémy s publikováním. Např. jsem zjistil, že mi v databázi chybí některá data nutná pro službu, v takovém případě jsem musel požádat kolegu vývojáře, aby mi chybějící data doplnil. Další chybu, na kterou jsem narazil byl špatně nastavený datasource.

Poté jsem mohl služby vypublikovat.

4.2 Konfigurace prostředí – Titanium

Po vypublikování služeb, jsem musel upravit nastavení aplikace Titanium tak, aby tyto nové služby začal používat.
Musel jsem se připojit do databáze a upravit konfigurační tabulku CONFIGURATIONS.

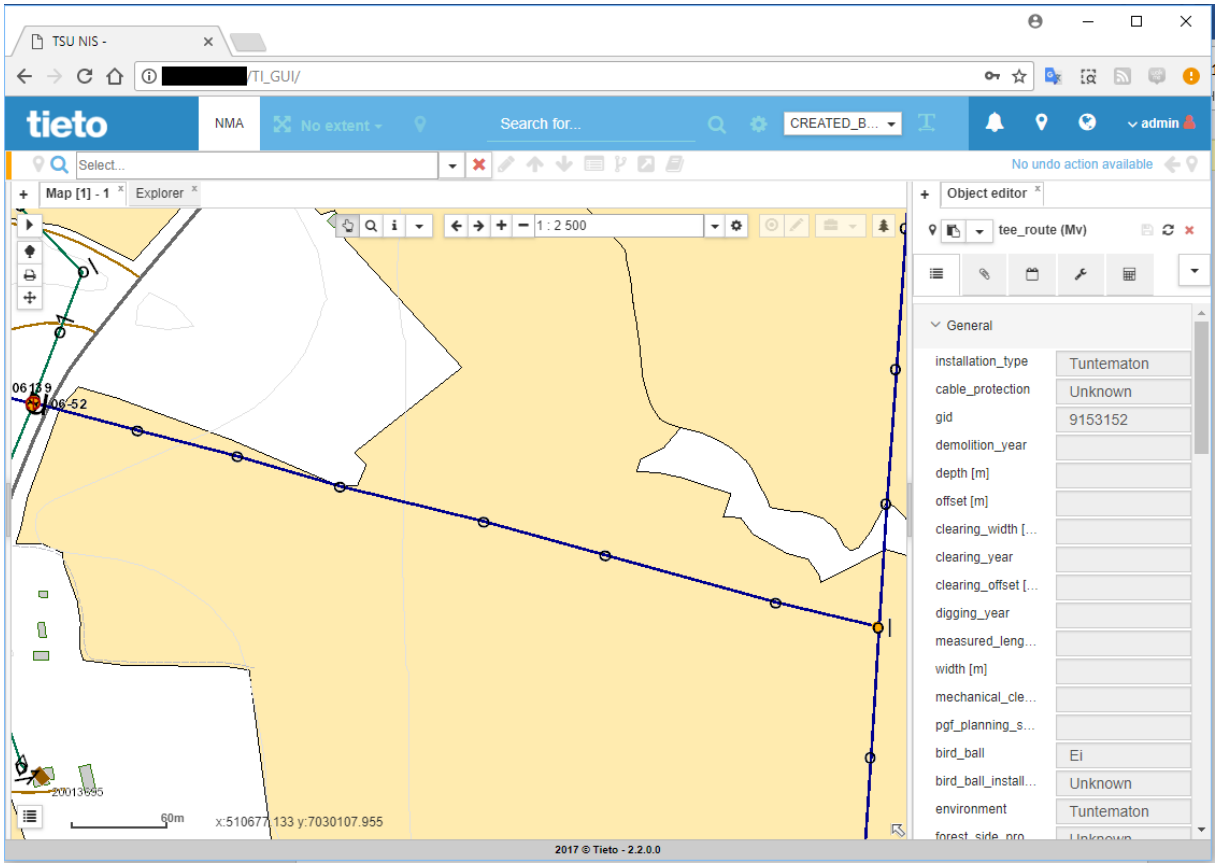


Obrázek 5: Tabulka CONFIGURATIONS

Stačilo každý řádek tabulky otevřít a nahradit výskyty řetězce **„TI_AUTO“** za **„TI_GUI“** a změnit IP adresu ArcGIS serveru. Je to následkem toho, že DB byla opět zkopírovaná se starého serveru.

Data v tabulce jsou uložena ve formě XML dokumentu, které jsou místy velmi dlouhé, naštěstí program HeidiSQL poskytuje nástroj na hromadně přejmenování řetězců, tudíž jsem nebyl nucen vytvářet skript na přejmenování.

Po spravení konfigurace, stačilo otevřít IIS, provést restart aplikace Titanium a zkusit se připojit.



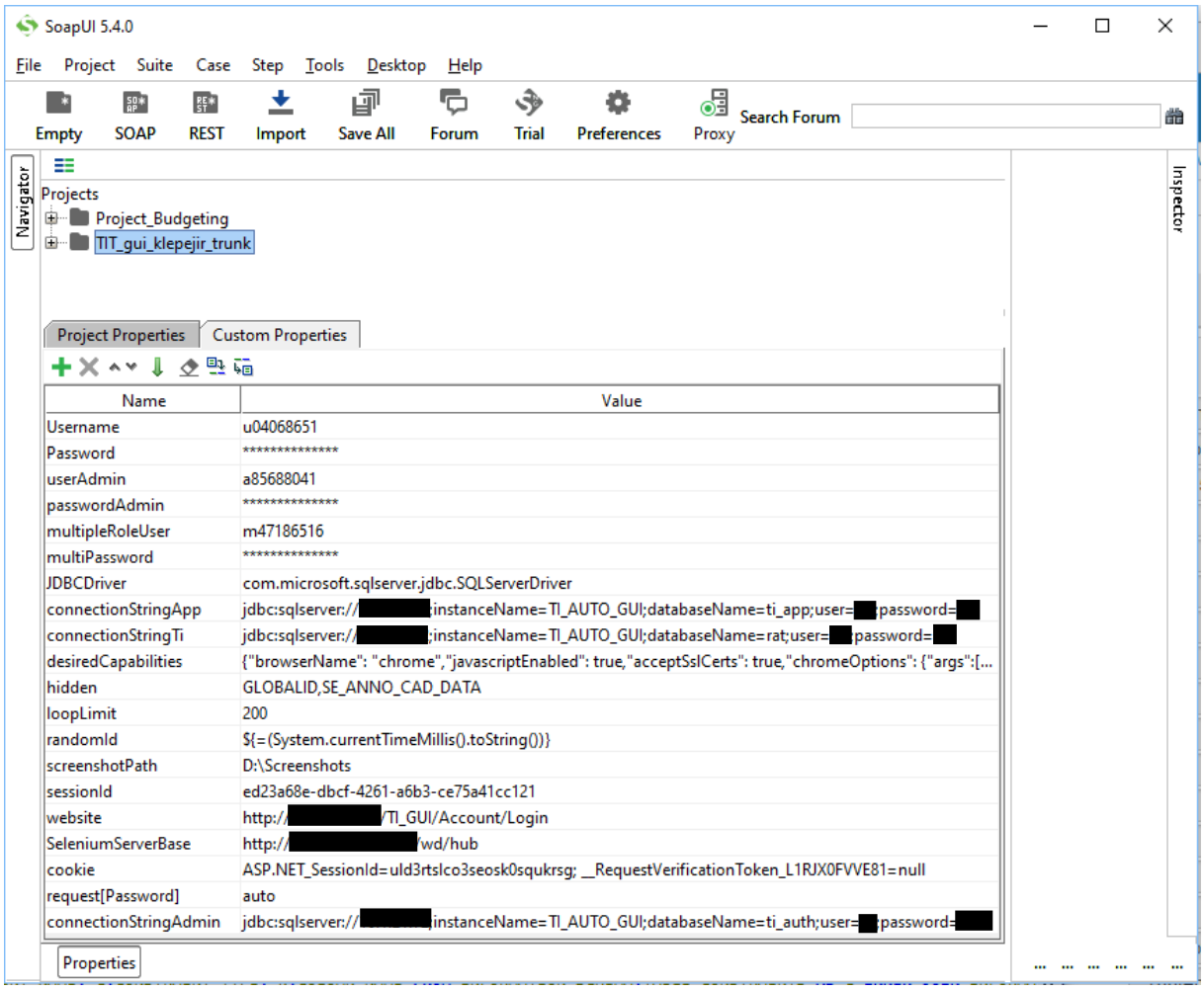
Obrázek 6: Titanium

Titanium se načetlo a po kontrole, zda se mapa vykresluje, zda lze na mapu kreslit a jestli funguje editace prvků, jsem mohl přejít k dalšímu úkolu, a to upravení GUI testů na nové prostředí.

Ostatní konfigurace, jako nastavení IIS a nastavení automatických build sestavení, jsem už nemusel řešit, protože byly už připravené od vývojářů.

4.3 Konfigurace GUI testů

Veškerá konfigurace se prováděla v programu SoapUI,
který slouží k testování REST a SOAP služeb [4], ale lze jej využít i pro GUI testování.



Obrázek 7: SoapUI – nastavení properties

Testy určené pro Titanium jsou uložené v projektu TIT_gui_klepejir_trunk, každý projekt má své properties, které fungují jako běžné proměnné. Testy při vykonávání využívají tyto properties, aby věděly, kde se mají vykonat a kdo je bude vykonávat.

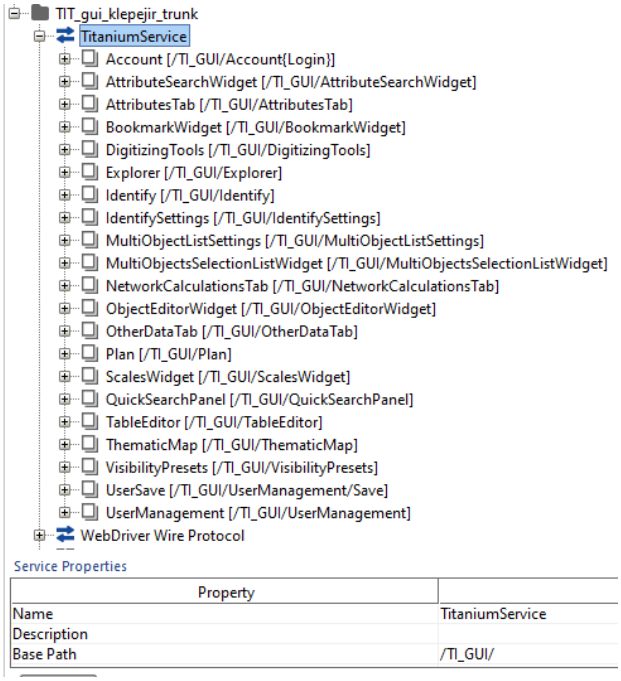
Properties, které byly nutné změnit.

- **website**
- **connectionStringApp**
- **connectionStringTi**
- **connectionStringAdmin**

Property website obsahuje webovou adresu aplikace Titanium, vůči kterému se testy pouští, bylo nutné ji nastavit na novou adresu.

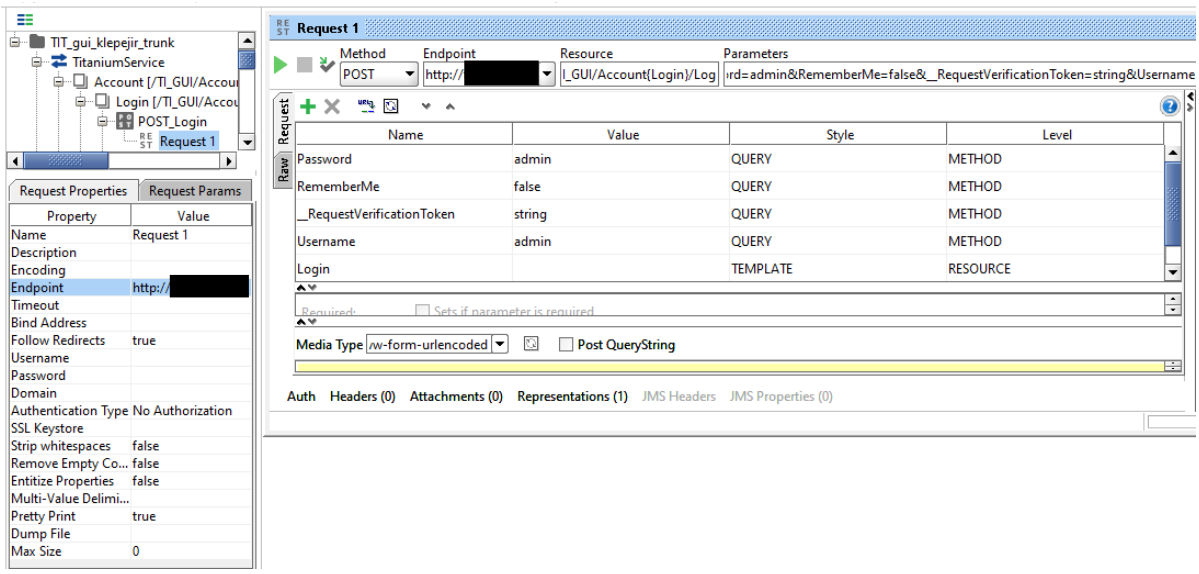
Další properties slouží k připojení do databáze aplikace Titanium, každá connection se připojuje do databáze jako jiný uživatel, a tudíž má k dispozici jiné tabulky. Musel jsem je upravit tak, aby obsahovaly IP adresu nového databázového serveru.

Dále jsem musel upravit samotné REST požadavky, které GUI testy v rámci projektu využívají. Tyto požadavky se nacházejí v TitaniumService a je jich poměrně velké množství.



Obrázek 8: SoapUI – REST požadavky

Opět bylo nutné opravit další property, konkrétně Base Path, která obsahovala TI_AUTO. Změnil jsem ji na TI_GUI, kvůli změně webové adresy aplikace Titanium. Sice REST požadavky už měly správný Base Path, problém ale byl, že jelikož se změnilo i DNS jméno, na kterém nové Titanium běží, musel jsem rozklikávat každý požadavek a měnit endpoint na správný.



Obrázek 9: SoapUI – nastavení endpoints

Bohužel v tomhle je SoapUI celkem hloupý software a nenabízí žádnou možnost, jak provést změnu endpoints u všech REST požadavků najednou, tedy kromě placené verze. [12]

Nyní jsem mohl GUI testy pustit vůči novému prostředí a zjistit, co vše je nutné opravit.

5 Oprava nefunkčních testů po migraci

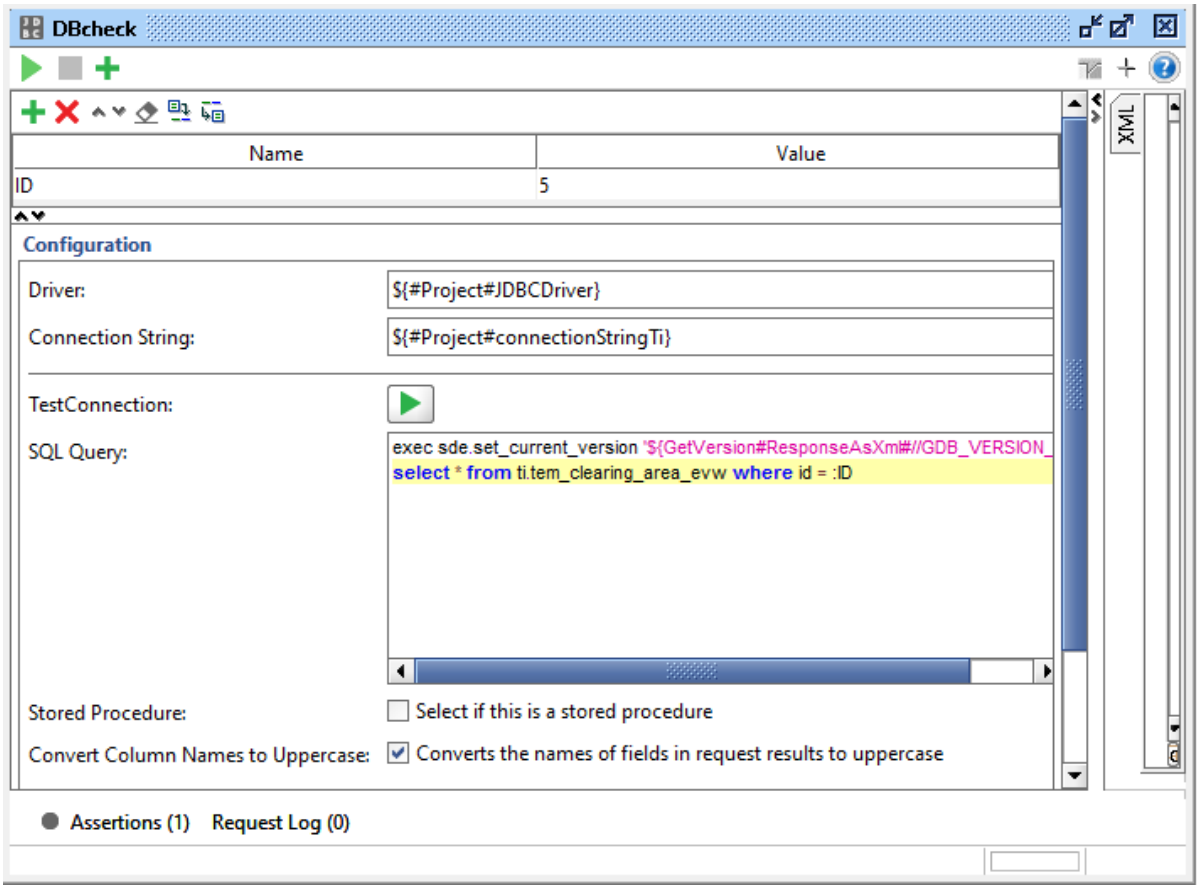
V době migrace bylo vytvořeno 226 testů, po jejich spuštění mi drtivá většina selhala. Důvody byly rozdílné, od změny prefixu tabulek databáze po chybějící data, se kterými GUI testy počítaly, protože je zanechávaly servisové testy.

5.1 Změna prefixu databáze

Při migraci se změnil prefix databáze aplikace Titanium, bohužel většina testů při provádění SQL příkazů psala explicitně prefix k tabulce i když to nebylo potřeba. Musel jsem tedy projít všechny testy, které pracovaly s databází a opravit je.

Původní prefix byl: **ti**, viz například tabulka **ti.tee_hv_pole**
Nový prefix je: **rat**, takže tabulka má nové celé jméno **rat.tee_hv_pole**

Oprava postižených testů byla jednoduchá, stačilo všude smazat explicitní prefix, který je zbytečný, protože se v testech používají různé connection strings do databáze, které samy o sobě prefix obsahují.



Obrázek 10: SoapUI – úprava DB prefixu

Zde vidíme příklad jednoho kroku testu, který má za úkol zkontrolovat, zda se vložená data opravdu vyskytují v databázi.

Konkrétně nám jde o řádek, kde se provádí select z tabulky tem_clearing_area_evw. Dotaz má u tabulky specifikovaný prefix, který je třeba smazat.

Connection String obsahuje:
jdbc:sqlserver://DB_IP;instanceName=TI_AUTO_GUI;databaseName=rat;user=DB_user;password=DB_password

Je vidět, že prefix u tabulek nepotřebujeme, protože je zvolen už v parametru databaseName.

Tímto způsobem jsem musel opravit 100 testů, tedy téměř polovinu z celkového množství.

Mohl jsem tedy přejít k dalšímu bolavému místu, a to závislost GUI testů na servisové.

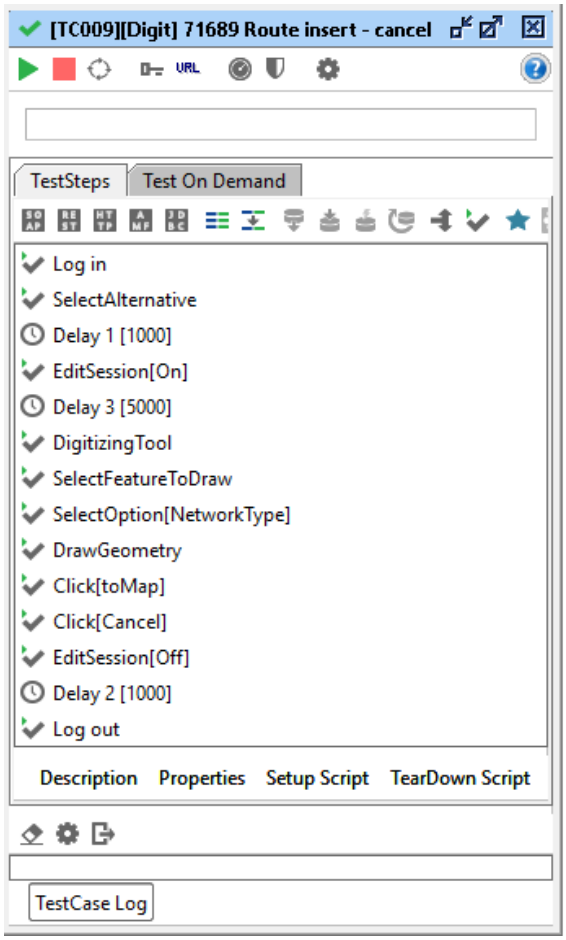
5.2 Oprava závislosti GUI testů na servisových testech

Tato část oprav byla asi nejnáročnější, velmi velké množství testů spoléhalo na data, která byla dodaná servisovými testy, jakmile se ale udělala migrace na nové samostatné Titanium, kde se servisové testy nebudou pouštět, nastal problém. Některé testy šly snadno opravit, u většiny to ale bylo složité. Někde byla závislost testů tak velká, že selhával celý Test Suite.

Test Suite je soubor testů, které spolu tematicky souvisejí.

Příkladem takového Test Suit je Digitizing, což jsou testy, které testují kreslení do mapy a funkčnost mapových prvků, které se mají vykreslit.

Máme zde příklad jednoho testu ze zmíněného Digitizing, vidíme, že každý test se skládá z jednotlivých kroků, některé kroky jsou akorát „moduly“, které samy o sobě obsahují další kroky a ty kroky mohou být opět moduly či naopak „nízko úrovňové“ příkazy, které pocházejí ze Selenium Frameworku.

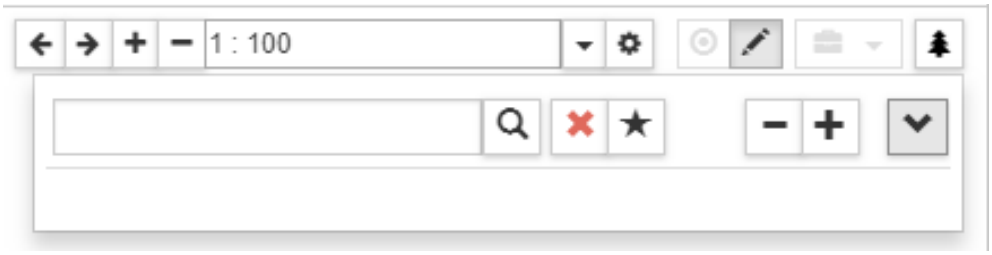


Obrázek 11: SoapUI – TC009[Digit]

Jako první krok je přihlášení do aplikace, poté výběr alternativy, tedy jednoduše řečeno zvolení, do jaké mapy chceme kreslit.

Následuje Delay 1, zapnutí EditSession, což znamená povolení úprav mapy a další delay.

Poté kliknutí na tlačítko Digitizing, které otevře malé okno obsahující prvky ke kreslení a máme problém. Viz obrázek číslo 12.



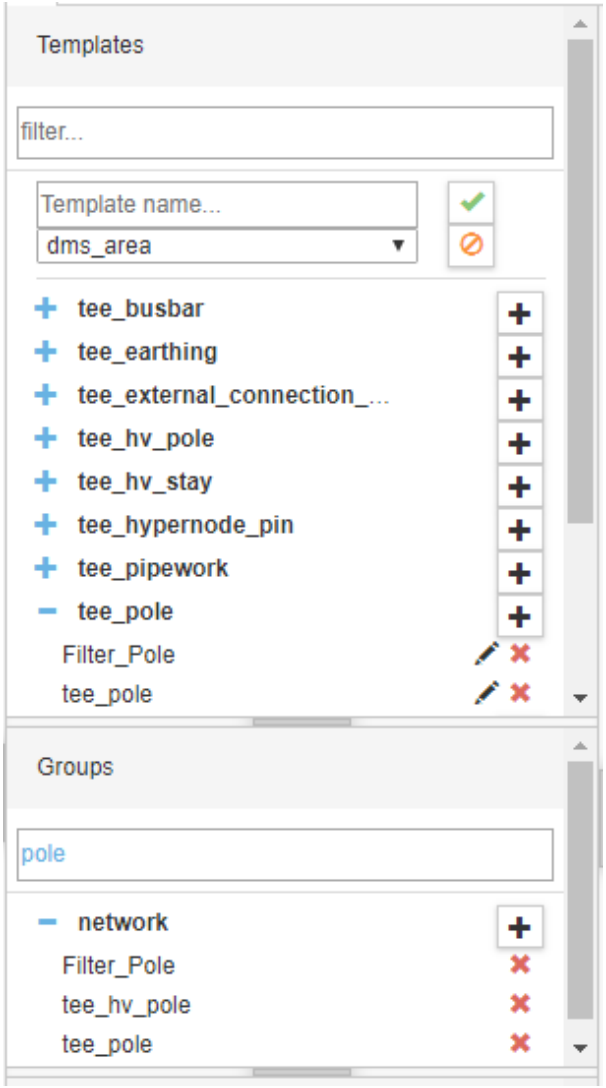
Obrázek 12: Titanium – digitizing

Okno neobsahuje žádné prvky ke kreslení, a to je příklad testu, který spoléhal na data, o které si myslel, že je bude mít vždy k dispozici, protože byly dodány servisovými testy.

Tento problém postihoval celý Digitizing. Musel jsem tedy vymyslet, jak data dodat a kde.

Využil jsem možnosti tzv. Setup skriptu, což jsou příkazy, které se provedou v rámci Test Suit jednou, a ještě předtím, než se spustí jakýkoli test.

Ideální místo na provádění různých předpříprav, zbývalo jenom vytvořit test, který se postará o dodání samotných dat.



Obrázek 13: Titanium – templates

Ještě, než přejdu k samotnému testu, vysvětlím, jak funguje v aplikaci Titanium kreslení mapových prvků do mapy. V prve řadě se těmto prvků v terminologii aplikace Titanium říká features. Každá feature obsahuje různé vlastnosti, které lze ještě před kreslením vyplnit, např. u stožáru můžeme určit, pro jaké elektrické napětí je určený. Z tohoto důvodu lze vytvářet takzvané templates, ve kterých lze ke každé feature nastavit předdefinované atributy. Tyto templates je třeba dále přesunout do groups, které říkají, zda se má template použít pro mapu či pro jinou část aplikace Titanium.

Tedy pokud nějaký test počítal, že po kliknutí na tlačítko Digitizing najde například feature tee_pole, což je ten stožár, musím mít k feature tee_pole vytvořený template, který se může jmenovat stejně a tato template musí být přidána do group, která umožnuje kreslení na mapu, takovou group je network.

Pokud tedy chci zařídit, aby testy neselhaly v případě chybějící feature v Digitizing, musím zařídit, aby se všechny templates vytvořily a správně zařadily do groups.

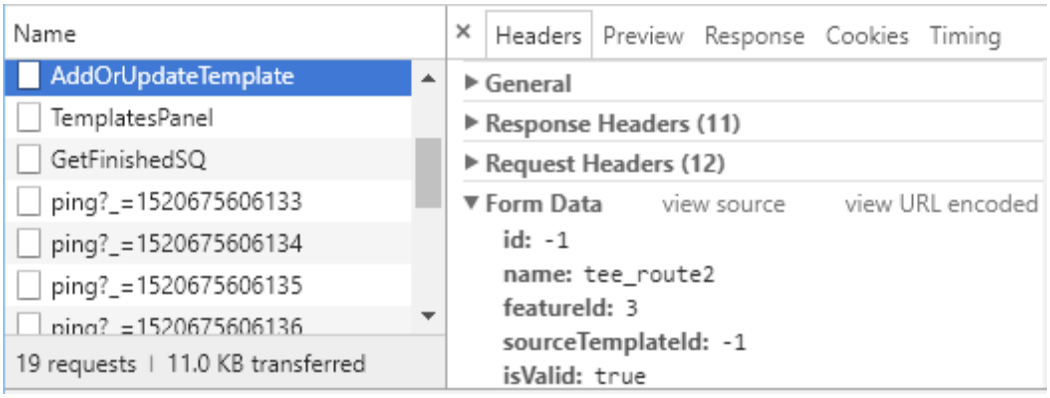
Možná se objevuje otázka, proč kvůli tomu musím psát nějaký test? Stačí to jednou ručně nastavit a je hotovo. Jde o to, že se každou noc databáze smaže, aby se testy pouštěly vůči čistému prostředí.

Přejdu tedy k samotnému testu.

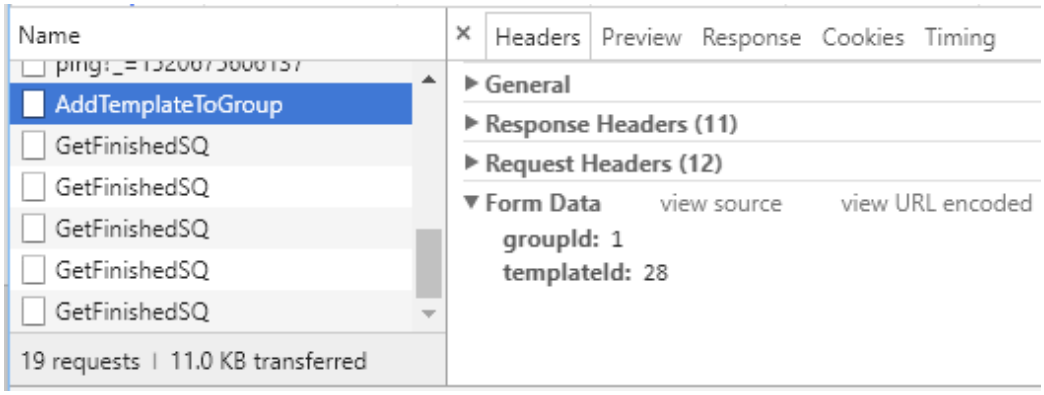
Abych mohl vytvořit test řešící tento problém, věděl jsem, že rozhodně nebude jako ostatní GUI testy, to znamená, že by se templates nevytvářely tak, jak je vytváří uživatel přes GUI aplikace Titanium, protože by to bylo za prvé pomalé a za druhé zbytečné, protože se nejedná o běžný test, ale jen o jakousi přípravu prostředí pro opravdové testy z Test Suit Digitizing.

Musel jsem tedy zjistit REST požadavky, které se posílají aplikaci Titanium při vytváření templates a při jejich zařazování do skupin.

Zjištění bylo jednoduché, stačilo akorát použít Chrome Develeper Tools.



Obrázek 14: Chrome Developer Tools – odchycení REST požadavků



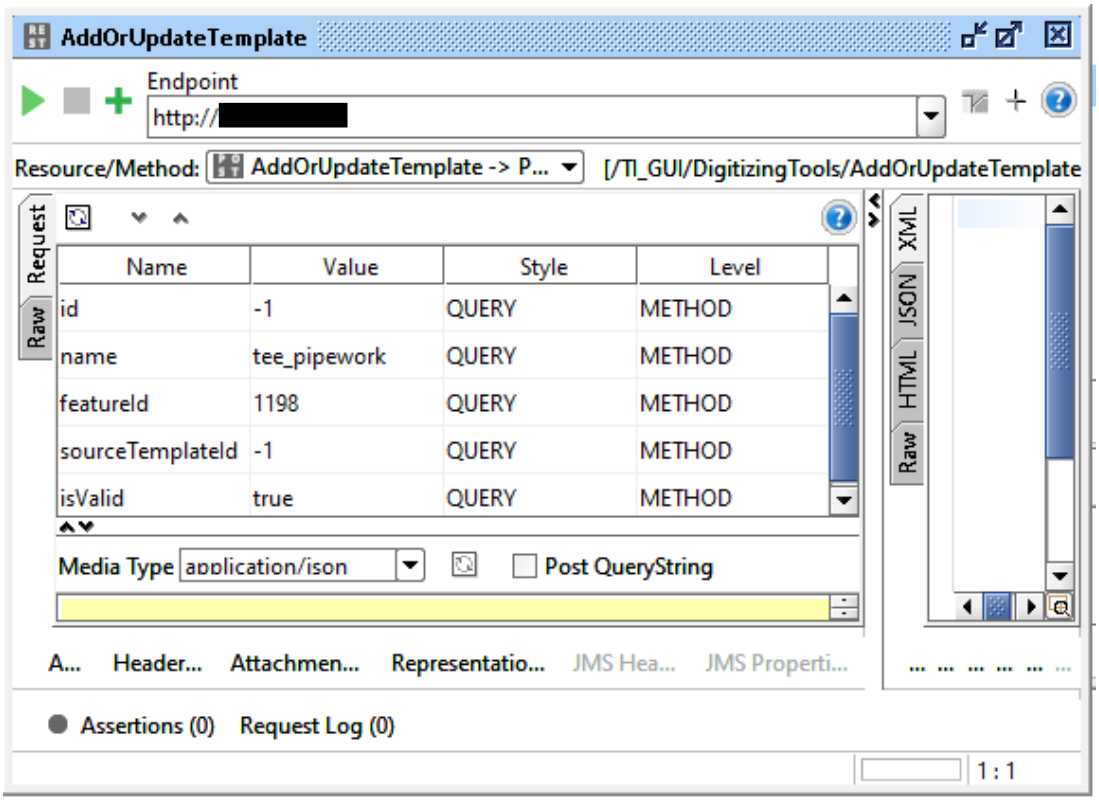
Obrázek 15: Chrome Developer Tools - REST požadavek AddTemplateToGroup

Nyní vím, že musím použít REST požadavek AddOrUpdateTemplate a AddTemplateToGroup.

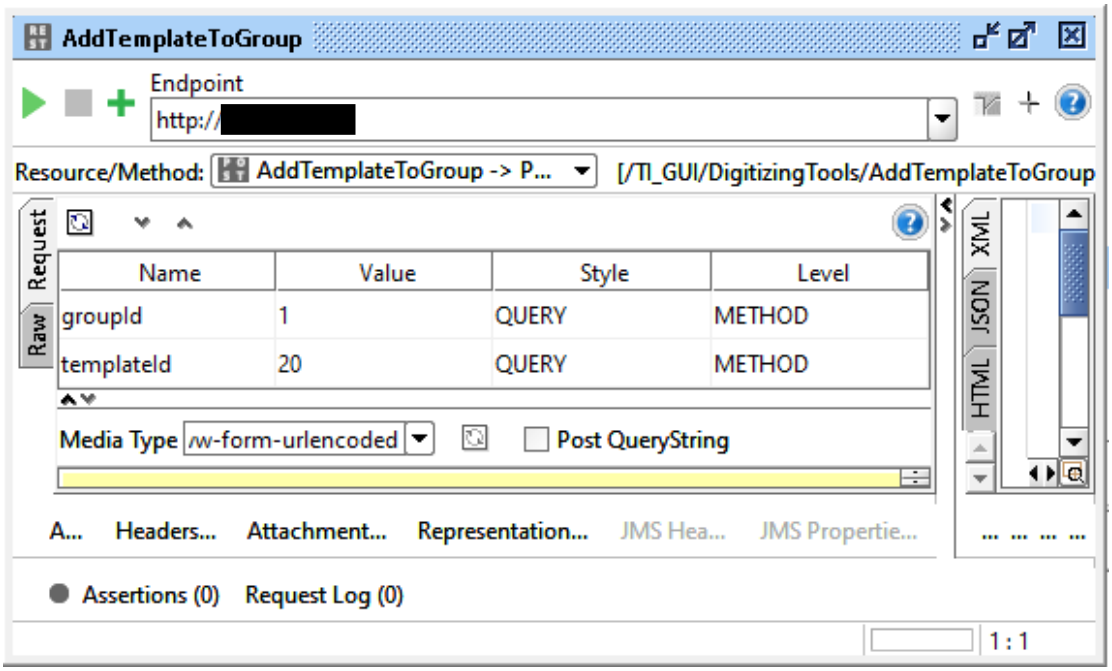
Těmto požadavkům musím předat patřičná Form Data:

- **id**: -1 – zde bude vždy -1, protože když se posílá REST na vytvoření template, nevíme, jaké id bude mít.
- **name**: tee_route2 – pojmenování template.
- **featureid**: 3 – jedná se o ID feature, ke které se vytváří template.
- **sourceTemplateId**: -1 – nezjistil jsem, k čemu tato property slouží, ale při mých pokusech byla vždy stejně nastavená, tedy není důležitá.
- **isValid**: true – nezjistil jsem, k čemu tato property slouží, ale při mých pokusech byla vždy stejně nastavená, tedy není důležitá.
- **groupId**: 1 – jde o ID group, ke které se template přiřadí, v tom to případě je to network.
- **templateId**: 28 – opět další ID, konkrétně jde o ID template, která se má přiřadit ke specifikované group.

V SoapUI jsem musel vytvořit tyto REST požadavky, abych je mohl ve svém testu použít:



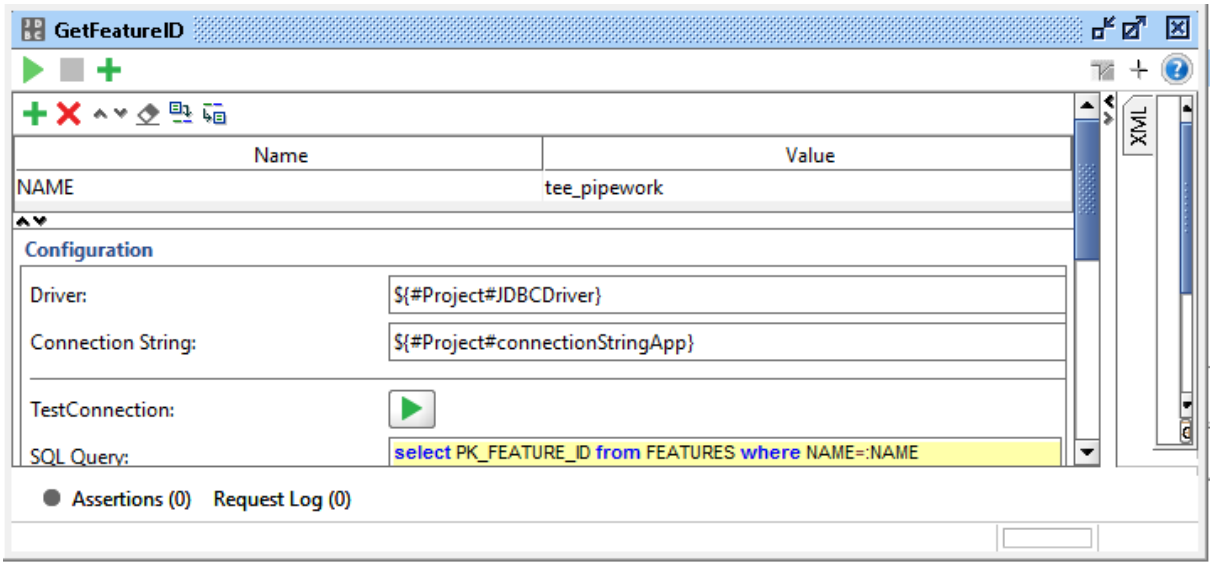
Obrázek 16: SoapUI - REST požadavek AddOrUpdateTemplate



Obrázek 17: SoapUI – REST požadavek AddTempateToGroup

Bohužel samotné REST požadavky mi nestačí, minimálně ještě potřebuji znát ID feature, ID group a ID template, abych je mohl použít. Tyto informace si musím vytáhnout z databáze, protože výše zmíněné REST požadavky dostanou jako odpověď pouze OK.

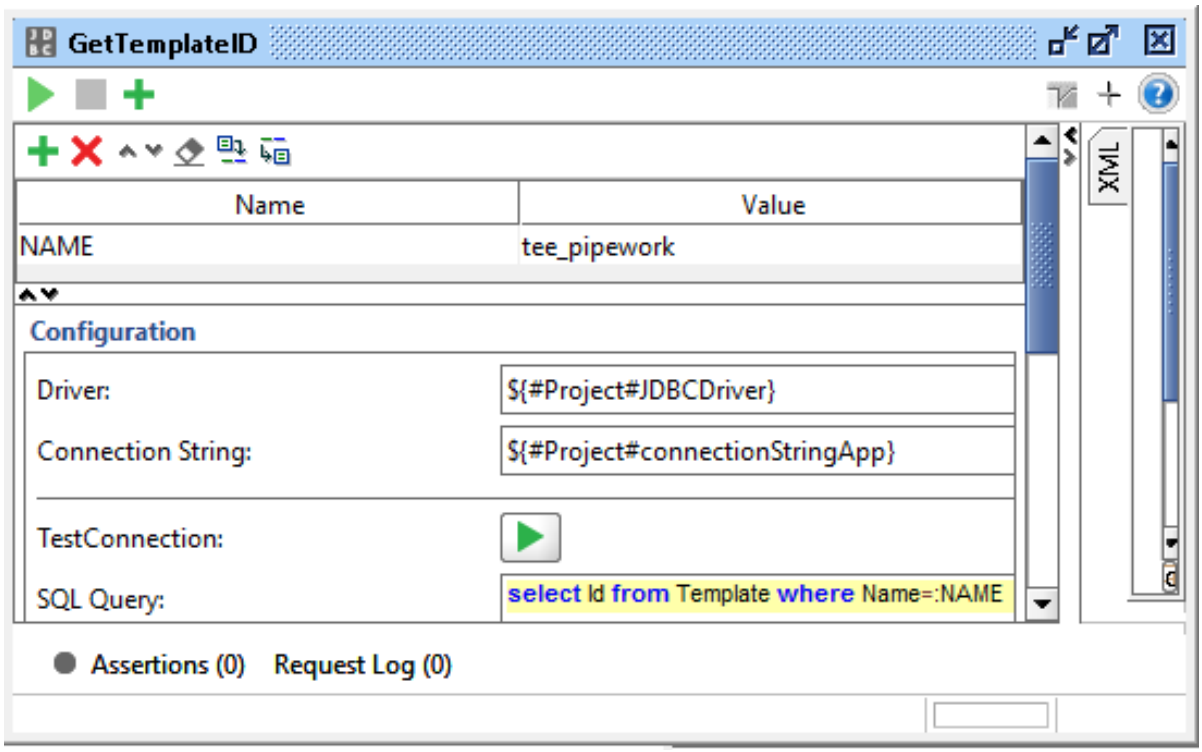
SQL dotaz na zjištění feature ID:



Obrázek 18: SoapUI – JDBC krok GetFeatureID

V dotazu je použít parametr, protože chci, aby SQL dotaz byl použitelný pro obecné features.

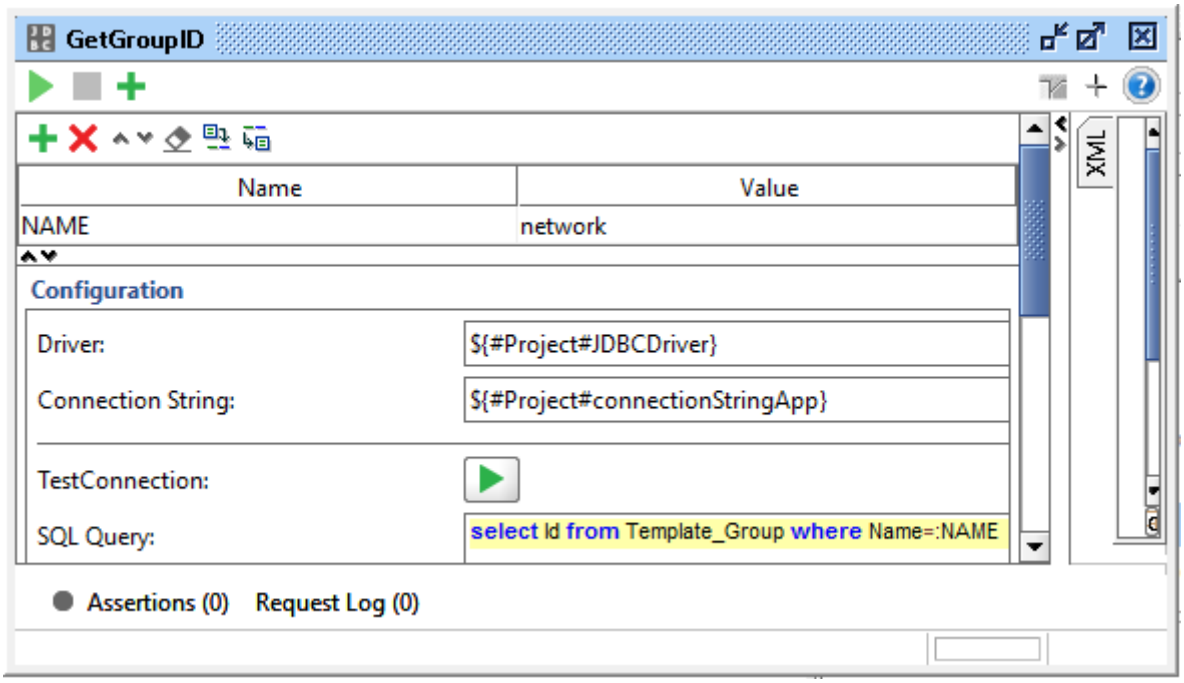
SQL dotaz na zjištění template ID:



Obrázek 19: JDBC krok – GetTemplateID

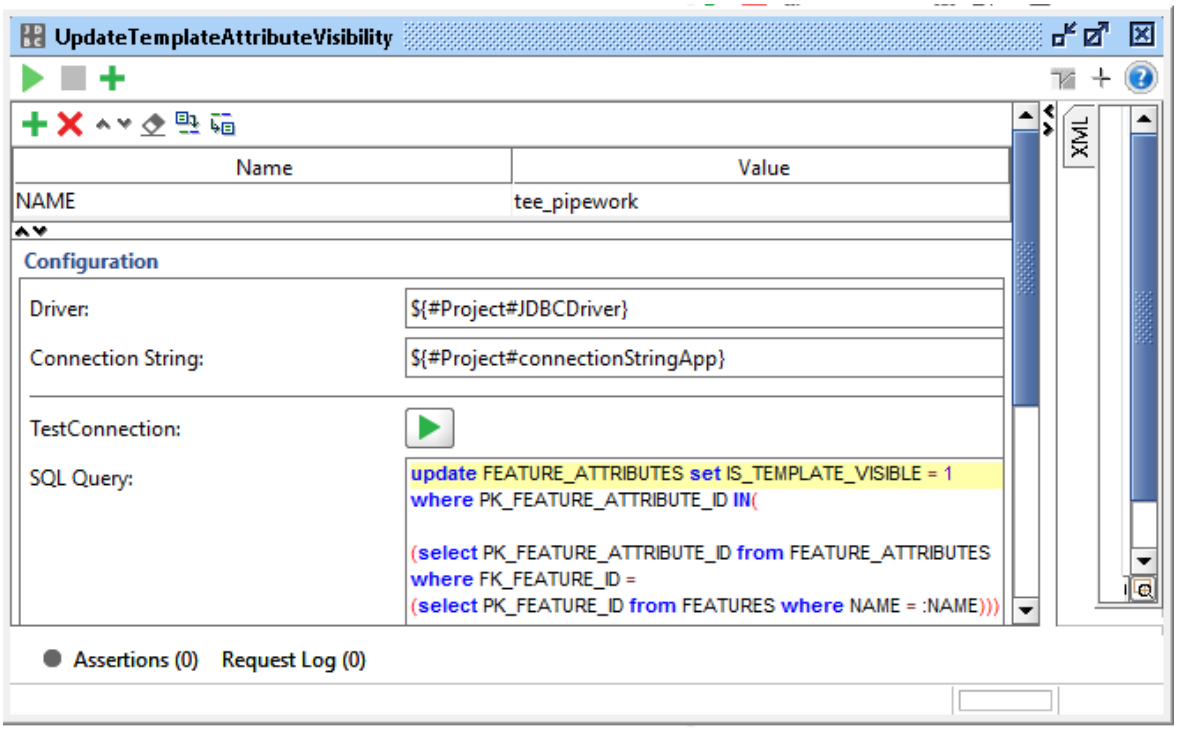
Opět parametrický, možná působí zmateně to, že je hodnota NAME u obou dotazu stejná. Protože template může mít stejné jméno jako feature, ke které se template vytváří.

SQL dotaz na zjištění group ID:



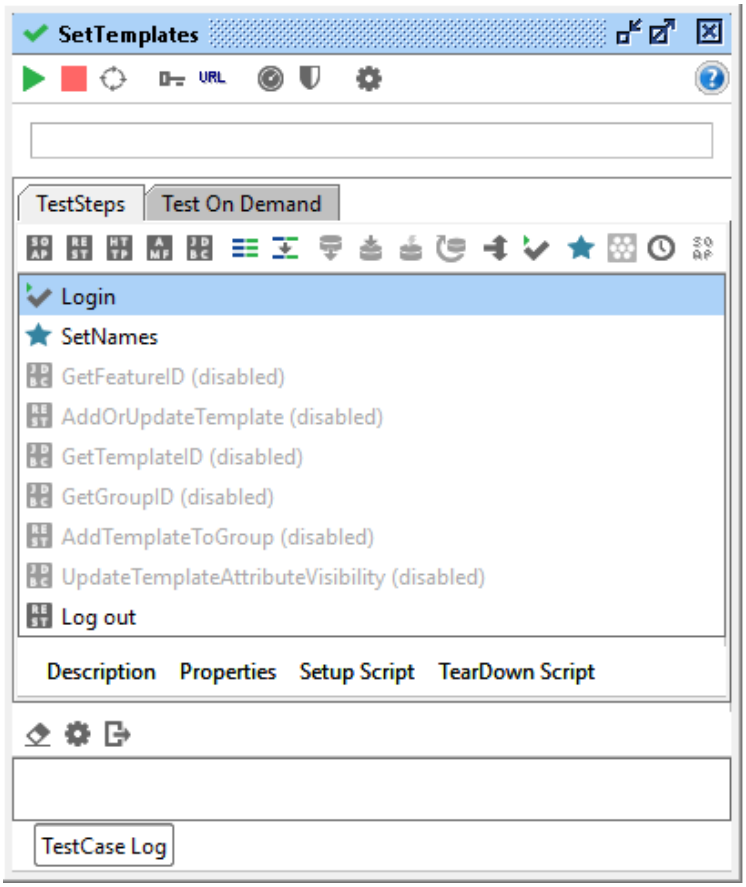
Obrázek 20: JDBC krok – GetGroupID

Téměř mám vše, co potřebuji, ale až na poslední věc. Jak jsem již zmiňoval, tak každá template má své předdefinované atributy, které lze různě vyplnit, některé atributy ale můžou být vypnuté či povinné. Testy počítají s tím, že jsou všechny atributy k dispozici, stačilo opět použít SQL příkaz:



Obrázek 21: JDBC krok – UpdateTemplateAttributeVisibility

Celý vytvořený test vypadá takto:

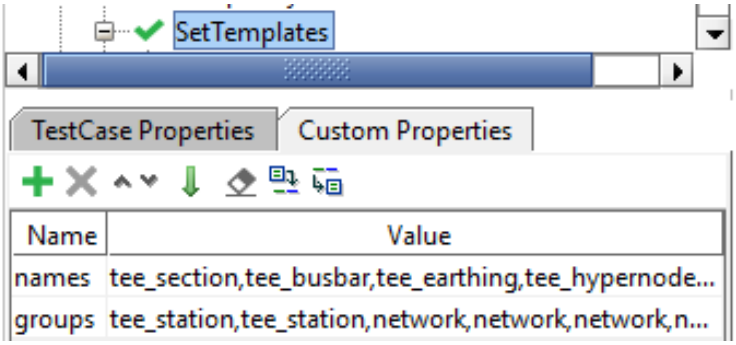


Obrázek 22: SoapUI – TC SetTemplates

Jako první se test přihlásí přes REST požadavek jako admin, aby mohl volat ostatní REST požadavky.

Poté následuje skript SetNames, který řídí vykonávání všech ostatních vypnutých kroků.

Skript je použitý z důvodu, že předávám testu do jeho property names seznam features, ke kterým se má vytvořit templates a do property groups, kde se má jaký template přiřadit.



Obrázek 23: SoapUI – TC SetTemplates properties

Test je tedy jednoduché rozšiřovat o nové templates.

Jako poslední krok se test odhlásí.

Samotný skript vypadá takto:

```
def names = context.expand( '${#TestCase#names}' )
def groups = context.expand( '${#TestCase#groups}' )
def values = names

values = names.split(',')
groups = groups.split(',')

def i = 0
for (val in values) {
    testRunner.testCase.getTestStepByName("GetFeatureID").setPropertyValue("NAME",val)
    testRunner.runTestStepByName("GetFeatureID");
    def featureId = context.expand( '${GetFeatureID#ResponseAsXml#//PK_FEATURE_ID}'  )

    testRunner.testCase.getTestStepByName("AddOrUpdateTemplate").setPropertyValue('name',val)
    testRunner.testCase.getTestStepByName("AddOrUpdateTemplate").setPropertyValue('featureId'
            ,featureId)
    testRunner.runTestStepByName("AddOrUpdateTemplate")

    testRunner.testCase.getTestStepByName("GetTemplateID").setPropertyValue("NAME",val)
    testRunner.runTestStepByName("GetTemplateID")
    def templateId = context.expand( '${GetTemplateID#ResponseAsXml#//ID}'  )

    def val2=groups[i++]
    testRunner.testCase.getTestStepByName("GetGroupID").setPropertyValue("NAME",val2)
    testRunner.runTestStepByName("GetGroupID")
    def groupId = context.expand( '${GetGroupID#ResponseAsXml#//ID}'  )

    testRunner.testCase.getTestStepByName("AddTemplateToGroup").setPropertyValue('templateId'
            ,templateId)
    testRunner.testCase.getTestStepByName("AddTemplateToGroup").setPropertyValue('groupId',groupId)
    testRunner.runTestStepByName("AddTemplateToGroup")
    Thread.sleep(1000)


testRunner.testCase.getTestStepByName("UpdateTemplateAttributeVisibility").setPropertyValue("NAME"
            ,val)
    testRunner.runTestStepByName("UpdateTemplateAttributeVisibility")
}
```

Výpis 1: SetNames

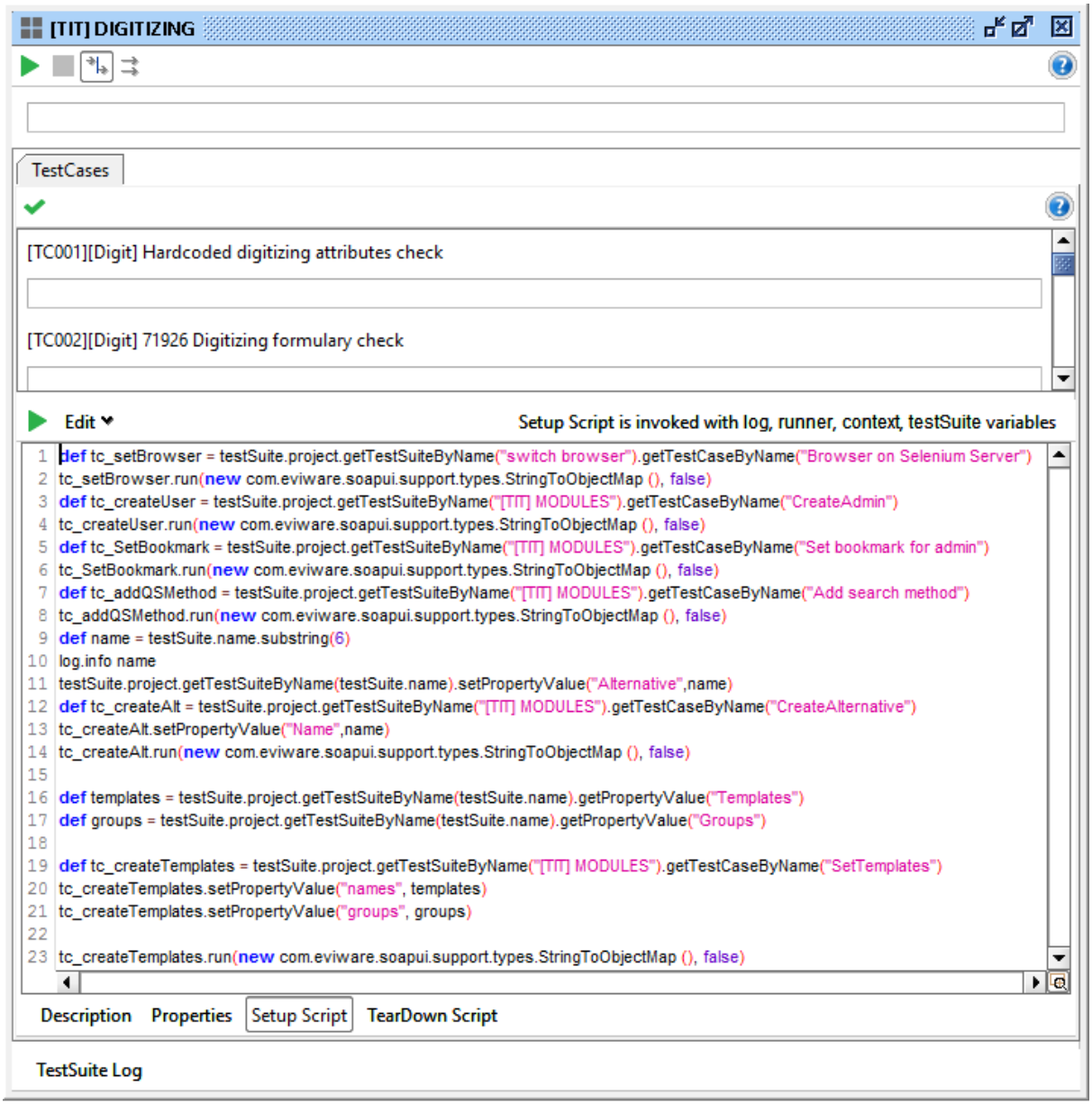
Na začátku skriptu dojde k rozbití obou řetězců names a groups na jednotlivé podřetězce, kde oddělovačem je čárka.

Poté se provádí FOREACH pro každý podřetězec v proměnné values. Ve smyčce dochází k zavolaní SQL dotazu na zjištění feature ID, pak se zavolá REST požadavek AddOrUpdateTemplate, kterému se předa požadovaná data.

Následuje zavolání obou SQL dotazů na zjištění template ID a group ID a předání obou ID REST požadavku AddTemplateToGroup.

Jako poslední příkaz ve smyčce se provede zavolání SQL dotazu na úpravu viditelnosti atributů.

Test je třeba ještě někde spouštět. Jelikož se musí provést jako první před jakýmkoli testem v Test Suit a právě jednou, musím umístit jeho spuštění do Setup skriptu, který tyto podmínky splňuje.



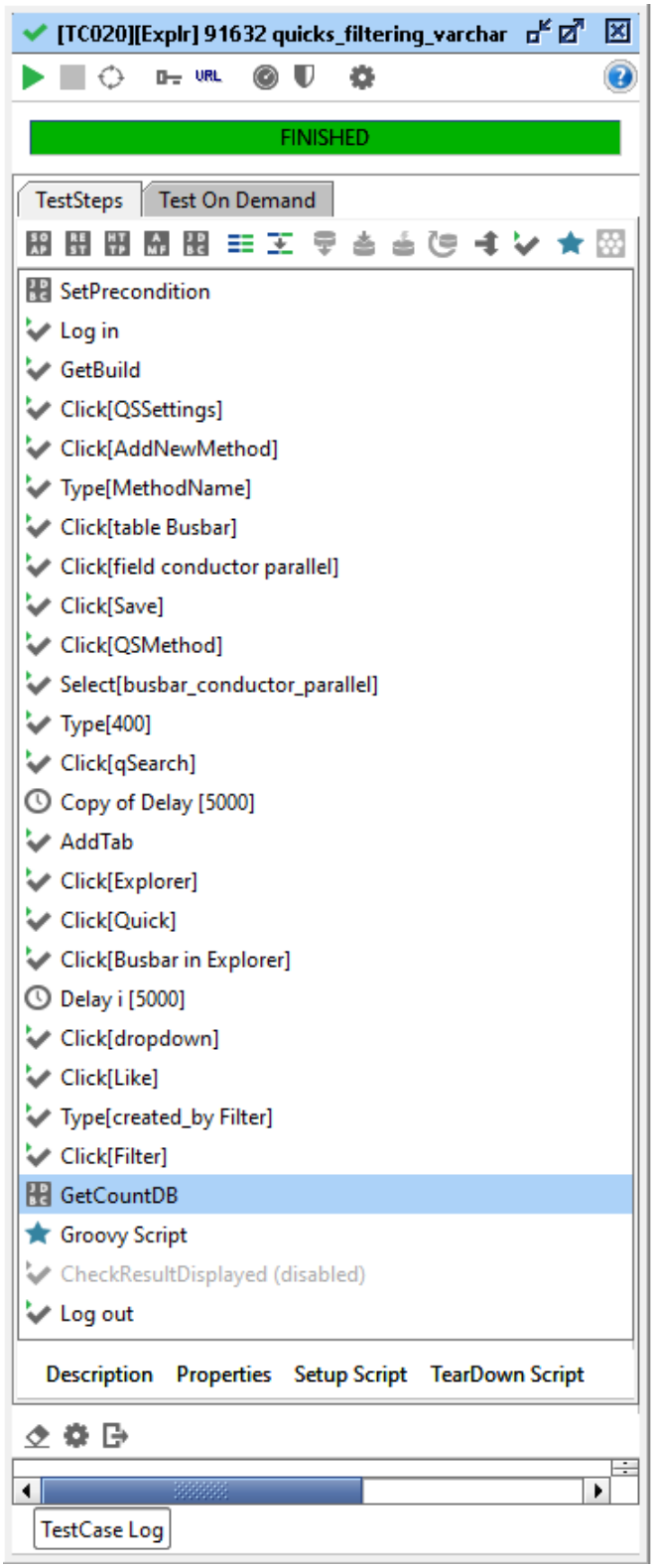
Obrázek 24: SoapUI – Setup Script

Zde byl příklad opravy testů spoléhajících na data dodaná servisovými testy, opravy byly nejnáročnější, tímto problémem bylo postihnuto 52 testů.

5.3 Oprava špatně napsaných testů

Kromě toho, že bylo mnoho testů závislých na jiné, jsem narazil také na testy, které selhávaly pouze z důvodu, že kontrolovaly data z GUI aplikace Titanium ne vůči databázi, ale vůči pevně zapsaných dat v testu.

Příklad takového testu může být tento:



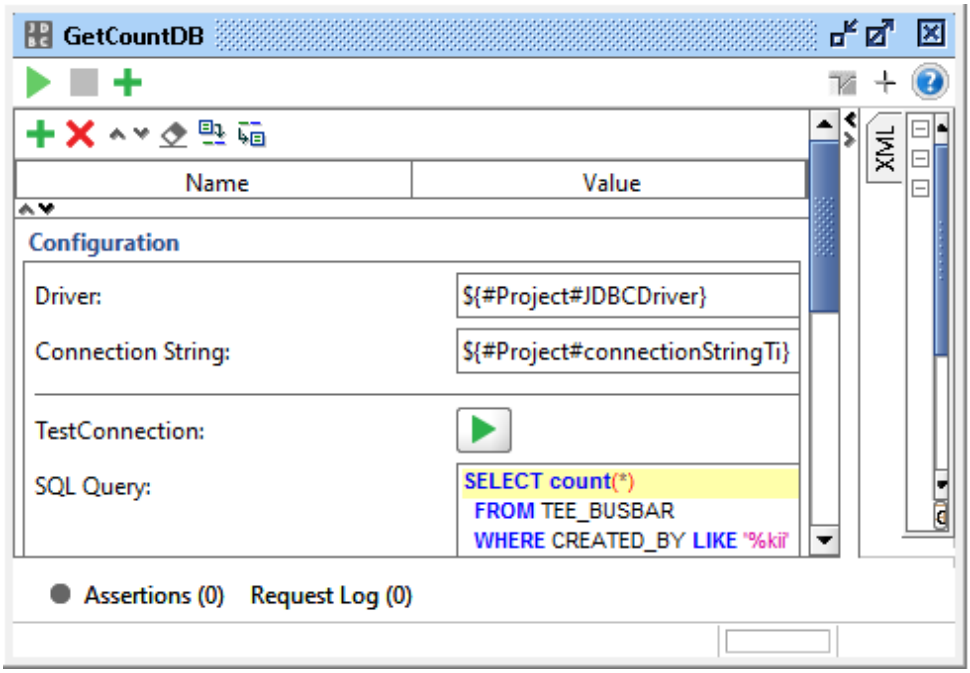
Obrázek 25: SoapUI – TC020[Explr]

Test má za úkol zkontrolovat, zda funguje filtrování features dle zadaných parametrů správně a jestli se vrací stejné množství výsledku jak v aplikaci Titanium, tak v databázi.

Problém ale je, že test vůbec nezískával počet výsledků z databáze, ale pouze jej získal z GUI aplikace Titanium, což je ještě v pořádku a poté porovnal vůči natvrdo zapsané hodnotě v testu.

Takový test je nejen zbytečný, ale také způsobuje problémy, změní-li se databáze, což byl můj případ.

Obrázek 28 je už z opraveného testu, kde na konci lze vidět krok GetCountDB, který tam předtím vůbec nebyl.



Obrázek 26: JDBC krok – GetCountDB

Ve skriptu níže vezmu hodnotu vracenou z databáze a připravím řetězec, který předám modulu CheckResultDisplayed, který provede finální otestování:

```
import com.eviware.soapui.model.testsuite.TestStepResult.TestStepStatus;
def count = context.expand( '${GetCountDB#ResponseAsXml#//_}'  );
def check = testRunner.testCase.getTestStepByName("CheckResultDisplayed");
def xpath = "//span[contains(.,'Showing 1 to 20 of " + count + " rows.')]";
check.setPropertyValue("IN-value",xpath);
testRunner.runTestStepByName("CheckResultDisplayed");
```

Výpis 2: Groovy Script TC020[Explr]

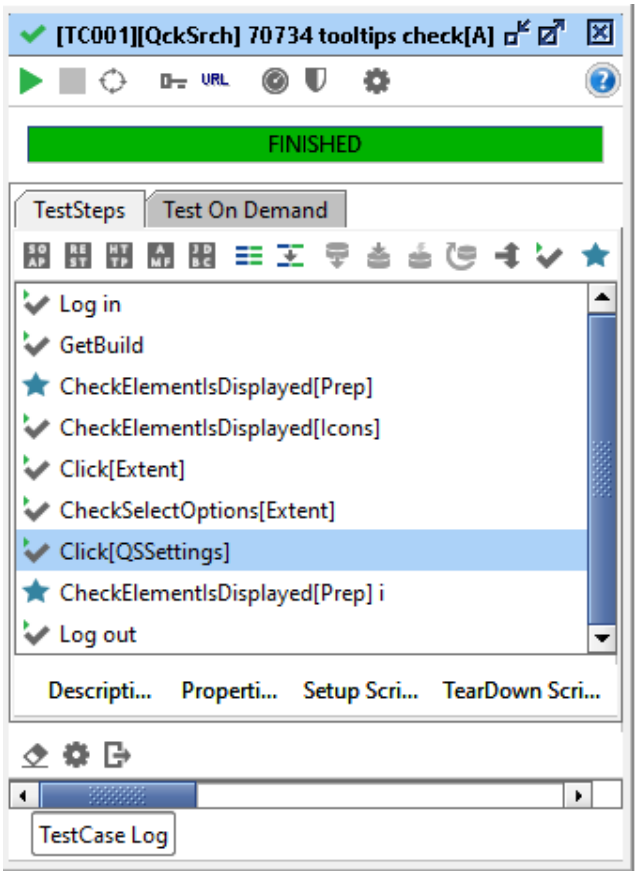
Původní skript obsahoval: ***def xpath = "//span[contains(.,'Showing 1 to 20 of 3992 rows.')]";***

Postihnuto touto vadou bylo kolem 23 testů.

5.4 Oprava selhávajících testů způsobených vývojem aplikace Titanium

Zbytek testů se nesl v duchu jednoduchých oprav, které spočívaly většinou jenom úpravou xpaths či SQL dotazů. Tyto selhávající testy selhávaly z důvodu vývoje aplikace Titanium, kdy se například změnilo jméno tabulky anebo se změnila struktura elementů v HTML stránce.

Tento test provádí kontrolu nastavení vyhledávacích metod, konkrétně kontroluje, zda nastavení obsahuje tlačítka a ikonky.



Obrázek 27: SoapUI – TC001[QckSrch]

Krok Click[QSSettings] ale nefungoval, protože jeho XPath vypadal:

```
//button[contains(@class,'SearchSettingsButton')]
```

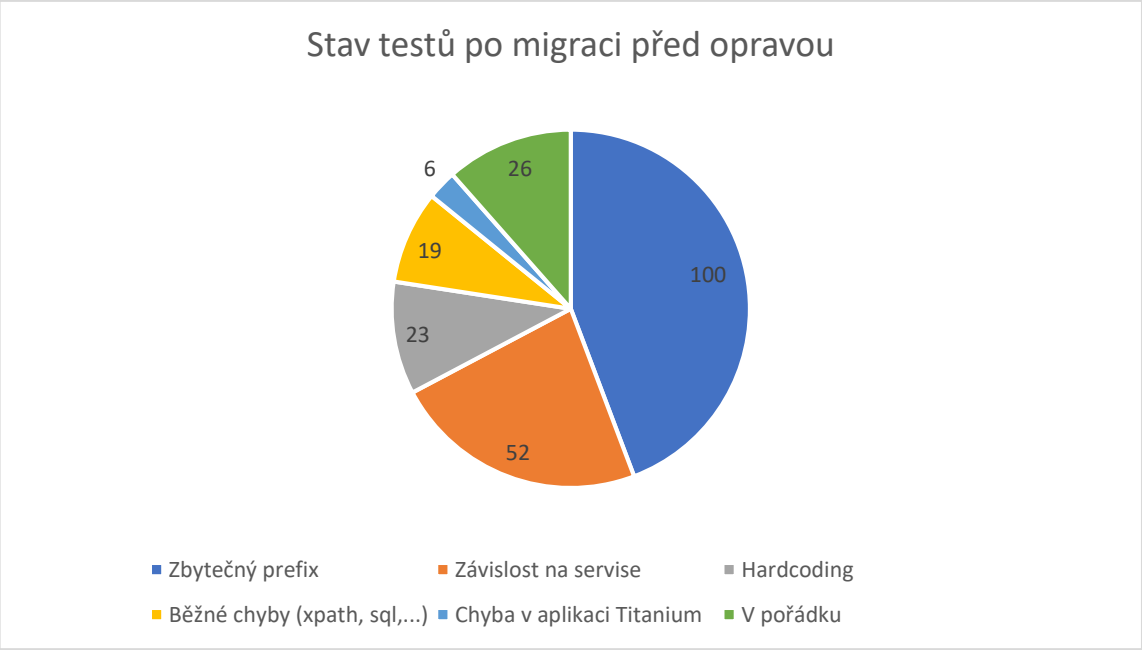
Došlo ale ke změně class, která se už nejmenovala jako SearchSettingsButton, ale quickSearchSettingsButton

Správný XPath vypadá tedy takto:

```
//button[contains(@class,'quickSearchSettingsButton')]
```

Poté test bezproblému proběhnul.

Postihnuto bylo 19 testů.



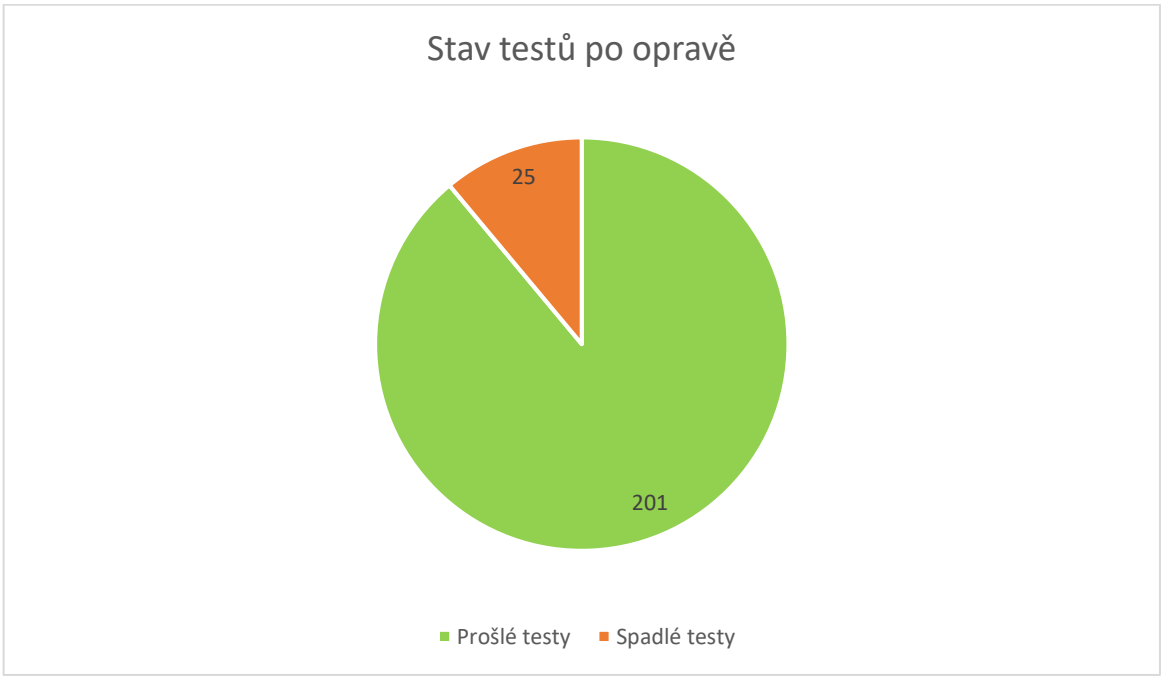
Graf 1: Stav testů po migraci před opravou

Některé testy obsahovaly více problémů, například zbytečný prefix a hardcoding, rozhodl jsem se, že budu příslušnost testu zařazovat dle problému, který jsem považoval za nejvážnější dle stupnice níže:

zbytečný prefix > závislost na servise > hardcoding > běžné chyby > chyba v aplikaci Titanium > v pořádku

6 testů selhávalo kvůli chybě v aplikaci Titanium, muselo se počkat, než vývojáři chyby opraví.

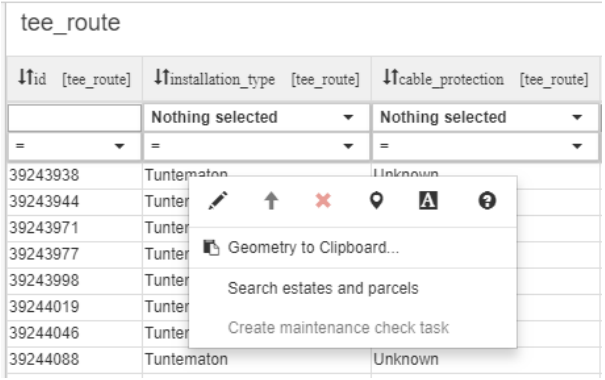
Opravě všech testů byla časově náročná, každopádně se mi jejich opravou podařila zvýšit jejich kvalita.



Graf 2: Stav testů po opravě

6 Vytvoření testů testující kontextovou nabídku

Dostal jsem úkol, kdy jsem měl vytvořit sadu testů testujících kontextovou nabídku aplikace Titanium, důvodem k těmto testům byla snaha ušetřit čas, které tyto testy zabíraly při manuálním testování.



Obrázek 28: Titanium – kontextová nabídka

6.1 Bezpečnost kontextové nabídky

Tyto testy, testující bezpečnost kontextové nabídky, testují, zda uživatel nemá přistup k položkám kontextové nabídky, ke kterým by přístup mít neměl. Než jsem se mohl pustit do psaní testů, musel jsem vytvořit pár podpůrných modulů.

Za prvé jsem musel nějak zajistit, aby test zjistil, že uživatel nemá přístup k položce kontextové nabídky, aniž by test selhal.

Za druhé se položky kontextové nabídky liší dle oprávnění, které uživatel má, takže jsem musel zajistit rychlé nastavování oprávnění uživateli.

Chtěl jsem pro kontrolu, zda uživatel nemá přístup ke konkrétní položce využít již hotové moduly „check element ARE or ARE NOT enabled“ a „check element ARE or ARE NOT displayed“.

Problém ale je, že tyto moduly spoléhají na to, že elementy identifikované pomocí xpaths na stránce existují a property IN-isEnabled je určena jenom pro zjištění, zda element je disabled či ne, ale ne pro zjištění, zda element vůbec na stránce je. Proto použití těchto modulů nedávalo smysl, protože by stejně hned selhaly. Musel jsem přijít na vlastní řešení.

Jako první nápad byl moduly prostě použít, ale při spouštění modulů ze skriptu bych odchytával výjimku, která se vyhazuje při nenalezení elementu. Bohužel tento postup nefungoval.

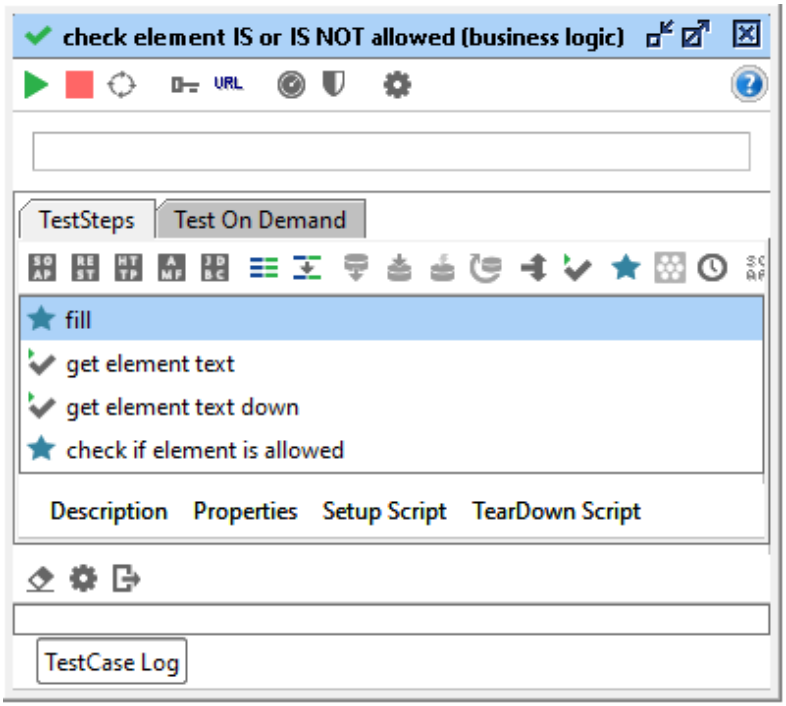
Napadla mě ale jiná věc, a to využít silných vlastností XPath technologie. Dočetl jsem se, že je možné dva naprosto nesouvisející xpaths spojit v jeden pomocí svislé čáry [13]. Například takto:

```
(//a[contains(@class,'locateUndoFeatureBtn')] | //footer)[1]
```

Oba xpaths jsou ještě obaleny kulatými závorkami a poté následuje [1] což znamená, že se má vybrat první nalezený XPath. Můj nápad spočívá v tom, že pomocí svislé čáry spojím první XPath, který je související s kontextovou nabídkou a druhý XPath, který bude sloužit jako „záchrana“ před selháním.

Tedy pokud položka locateUndoFeatureBtn se bude vyskytovat v kontextové nabídce, XPath vrátí jako první nalezenou hodnotu odkaz na položku. Pokud ale položka v kontextové nabidce nebude, dojde k nalezení patičky webu, která existuje vždy a vrátí se, tím pádem nedojde k selhání a já mám možnost jak zjistit, zda položka je či není v nabídce.

Vytvořil jsem tedy modul „check element IS or IS NOT allowed (business logic)“.



Obrázek 29: SoapUI - vlastní modul

V prvním kroku se vykoná skript, který zajistí správné vytvoření XPath, poté se spustí kroky get element text a get element text down, které zjistí, co XPath vrací.

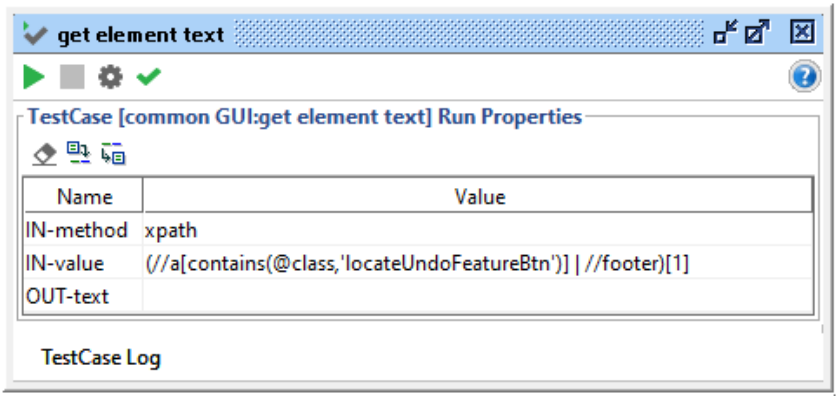
Následuje poslední skript, který vykoná samotnou kontrolu a dle nastavených properties zjistí, zda element existuje či ne a zda je to chyba anebo ne.

Skript fill:

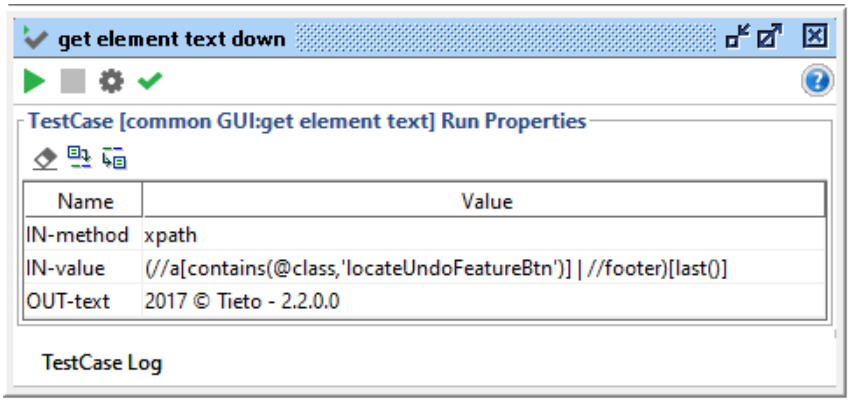
```
def getElementText = testRunner.testCase.getTestStepByName("get element text");
def getElementTextDown = testRunner.testCase.getTestStepByName("get element text down");
def value = context.expand( '${#TestCase#IN-value}' )
getElementText.setPropertyValue("IN-value", '(' + value +  ' | //footer)[1]');
getElementTextDown.setPropertyValue("IN-value", '(' + value +  ' | //footer)[last()]');
```

Výpis 3: Groovy skript fill

Kromě prvního formátu XPath s „[1]” jsem musel použít i jiný formát s „[last()]” na konci a to z důvodu, že některé elementy kontextové nabídky se nepochopitelné nacházejí až pod patičkou webu v HTML souboru.



Obrázek 30: krok get element text



Obrázek 31: krok get element text down

U obou modulů mě zajímá především property OUT-text, kterou dále využívám v posledním skriptu check if element is allowed:

```
def out = context.expand( '${get element text#OUT-text}' )
def out2 = context.expand( '${get element text down#OUT-text}' )
def up = context.expand( '${#TestCase#IN-isUp}' )
def allowed = context.expand( '${#TestCase#IN-isAllowed}' )
if (up == 'true') {
    if (out.contains('Tieto') && allowed == 'false') {
        log.info 'OK';
    }
    else if (out.contains('Tieto') && allowed == 'true') {
        log.info 'KO - element is missing';
        assert 1 == 0 : 'KO - element is missing';
    }
    else if (!out.contains('Tieto') && allowed == 'true') {
        log.info 'OK';
    }
    else if (!out.contains('Tieto') && allowed == 'false') {
        log.info 'KO - element is living';
        assert 1 == 0 : 'KO - element is living';
    }
} else {
    if (out2.contains('Tieto') && allowed == 'false') {
        log.info 'OK';
    }
    else if (out2.contains('Tieto') && allowed == 'true') {
        log.info 'KO - element is missing';
        assert 1 == 0 : 'KO - element is missing';
    }
```

```
    else if (!out2.contains('Tieto') && allowed == 'true') {
        log.info 'OK';
    }
    else if (!out2.contains('Tieto') && allowed == 'false') {
        log.info 'KO - element is living';
        assert 1 == 0 : 'KO - element is living';
    }
}
```

Výpis 4: Groovy skript check if element is allowed

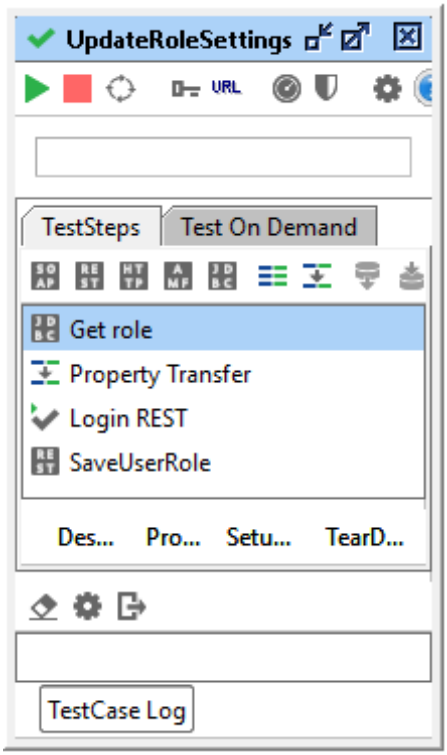
Skript v závislosti na hodnotě uložené v property isUp, která slouží k indikaci toho, zda je element nad patičkou webu, a v závislosti na property isAllowed, která zase říká, jestli je element povolený, provede kontrolu navracené hodnoty XPath a pokud nějaká podmínka selže, dojde k vyhození chyby a zastavení vykonávaní testu spolu s chybovou hláškou.

Tím jsem tedy vyřešil první problém, zbývalo ještě vytvořit moduly na rychlou změnu oprávnění uživatele.

Každý uživatel v aplikaci Titanium může být zařazen do mnoha rolí, kterým lze přiřazovat různá oprávnění, tudíž pokud má test otestovat kontextovou nabídku pro určitého uživatele, který drží různá oprávnění, ve skutečnosti je nutné tato oprávnění přiřadit roli, ve které je uživatel členem.

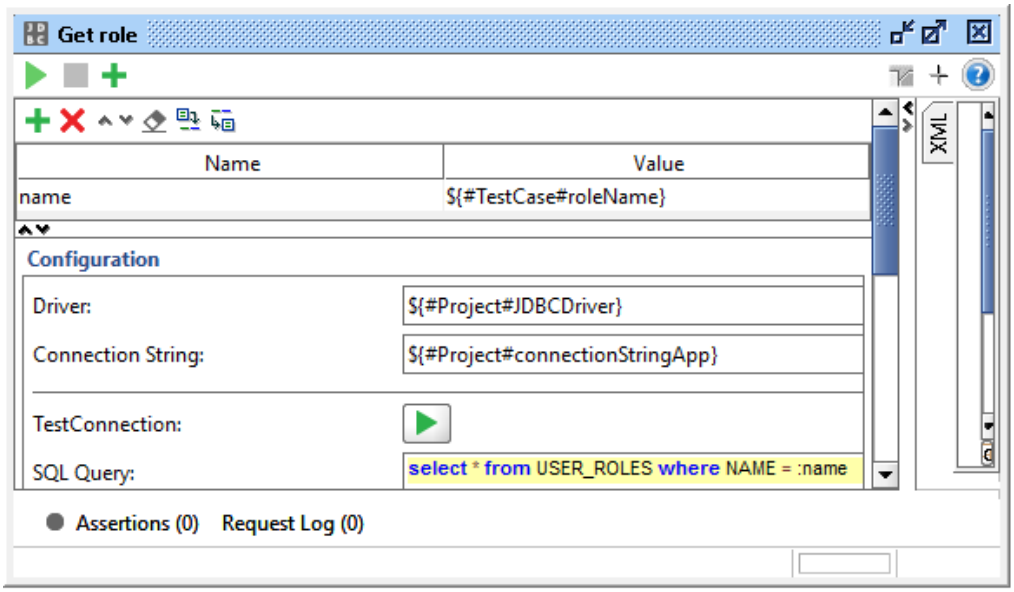
Bylo tedy třeba vytvořit modul, kterému když předám název role a seznam oprávnění, provede jejich přiřazení. Opět jsem nemohl přiřazování provádět přímo v GUI, protože by to trvalo hodně dlouho a testy na kontextovou nabídku jsou testy určené na otestovaní nabídky, ne na funkčnost přiřazování oprávnění roli.

Proto jsem vytvořil modul UpdateRoleSettings, který vše provede pomocí jednoho dotazu do databáze a jednoho zavolání REST služby. Modul jako properties přijímá název role a seznam oprávnění oddělených čárkou.



Obrázek 32: TC Update RoleSettings

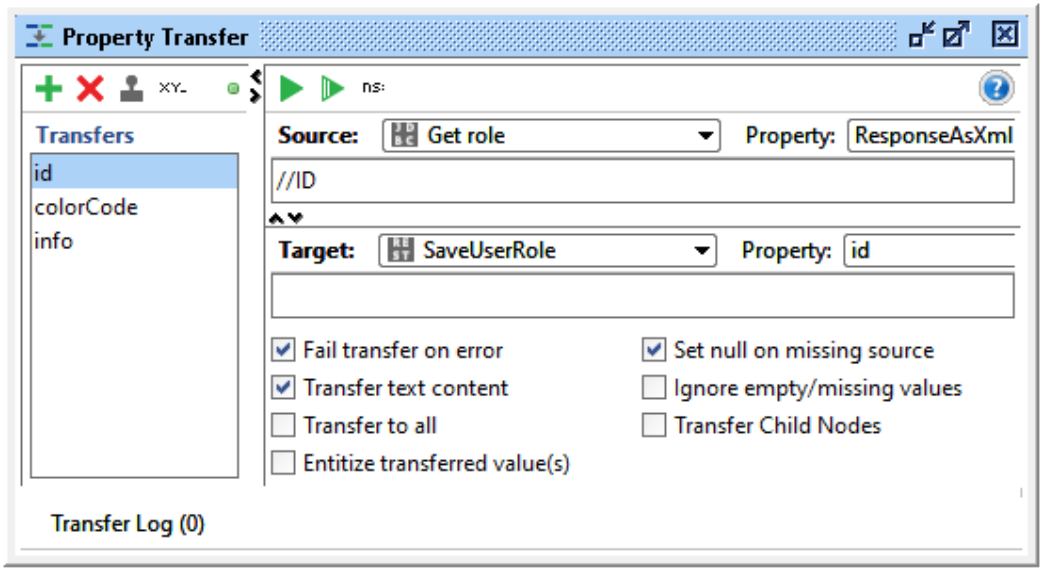
Jako první se provede SQL dotaz, který slouží ke zjištění ID role, kterou chceme upravit. ID je nutné, protože jej vyžaduje REST služba SaveUserRole. Dotaz je parametrizovaný.



Obrázek 33: JDBC krok Get role

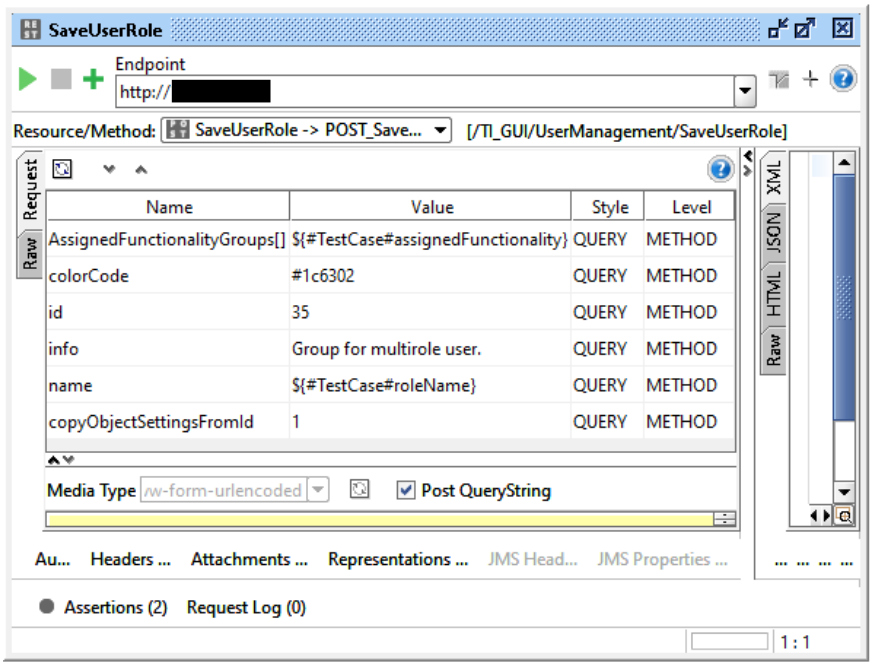
Po zjištění ID role se provede Property Transfer, jde o speciální krok v SoapUI, který umožnuje vzít výsledek jednoho kroku a poslat jej do jiného, v mém případě se ID pošle do REST služby SaveUserRole.

Kromě ID se pošle i colorCode a info.



Obrázek 34: Property Transfer

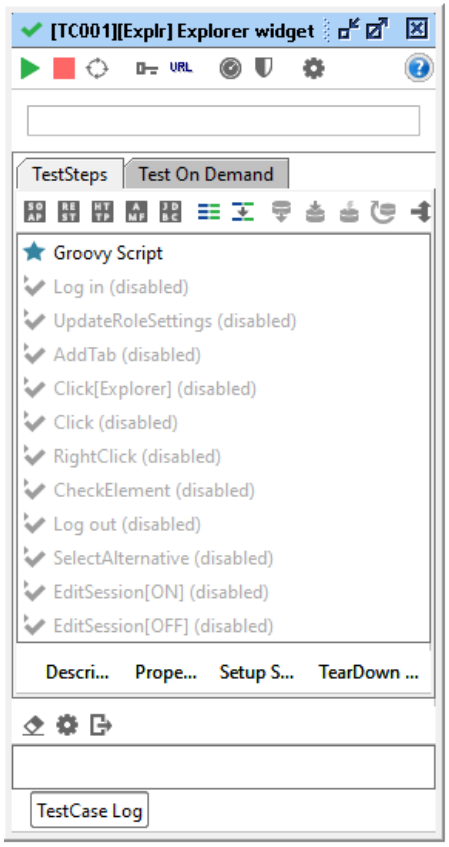
Krok Login REST provede přihlášení administrátora do aplikace, aby mohl později zavolat SaveUserRole.



Obrázek 35: REST požadavek SaveUserRole

Po zavolání se provedlo přiřazení oprávnění roli a uživatel může znovu vyvolat kontextovou nabídku, kde se zkontroluje, zda se v ní neobjevily položky, ke kterým nesmí mít přístup.

Jako příklad testu testující bezpečnost kontextové nabídky můžu uvést test níže:



Obrázek 36: SoapUI – TC001[Explr]

Jedná se o velmi složitý test, ve kterém se vykonává mnoho akcí, než dojde k finální kontrole nabídky. Test často volá různé kroky opakovaně, proto veškerou kontrolu nad tím, jak se kroky provádějí drží Groovy skript.

Kroky, které se budou používat, obsahují i modul UpdateRoleSettings.

Je nutné kontrolu provádět i při zapnuté EditSession, protože mění kontextovou nabídku.

Test je komplexní i v tom, že se uživatel bude často přihlašovat a odhlašovat, což není v GUI testech běžné. Důvodem k tomuto chování je skutečnost, že když se změní oprávnění role, ve které je uživatel členem. Nejsou nová pravidla aplikovaná okamžitě, ale až při příštím přihlášení.

Test má za úkol otestovat kontextovou nabídku v Explorer widget, což je část aplikace umožňující přístup k různým features a jejich správu.

Většina features má jinou kontextovou nabídku, test je tedy musí vzít v potaz.

Test kontroluje nabídku pro 9 různých nastavených oprávnění role.

Groovy skript:

```
String[] functionalGroups = ["Base", "Base,ApplicationSettings", "Base,Versioning",
"Base,Monitoring", "Base,AuthorityReporting", "Base,Versioning,NetworkCalculations",
"Base,Versioning,Digitizing", "Base,Versioning,Digitizing,PlanPosting",
"Base,Versioning,Digitizing,Maintenance"]
////////////////////////////////////////////////////////////////////////////////////////////////////
String[] contextButtons = ["//button[@data-action= 'openEditor']", "//button[@data-action=
'delete']", "//button[@data-action= 'multiEdit']"]
String[] contextButtonsAllowedBase = ["true", "false", "false"]
String[] contextButtonsAllowedApplicationSettings = ["true", "false", "false"]
String[] contextButtonsAllowedVersioning = ["true", "false", "false"]
String[] contextButtonsAllowedMonitoring = ["true", "false", "false"]
String[] contextButtonsAllowedAuthorityReporting = ["true", "false", "false"]
String[] contextButtonsAllowedNetworkCalculation = ["true", "false", "false"]
String[] contextButtonsAllowedDigitizing = ["true", "true", "true"]
String[] contextButtonsAllowedPlanPosting = ["true", "true", "true"]
String[] contextButtonsAllowedMaintenance = ["true", "true", "true"]

String[][] contextButtonsAllowed = [contextButtonsAllowedBase,
contextButtonsAllowedApplicationSettings, contextButtonsAllowedVersioning,
contextButtonsAllowedMonitoring, contextButtonsAllowedAuthorityReporting,
contextButtonsAllowedNetworkCalculation, contextButtonsAllowedDigitizing,
contextButtonsAllowedPlanPosting, contextButtonsAllowedMaintenance]
////////////////////////////////////////////////////////////////////////////////////////////////////
def checkElement = testRunner.testCase.getTestStepByName("CheckElement");
def rightClick = testRunner.testCase.getTestStepByName("RightClick");
def click = testRunner.testCase.getTestStepByName("Click");
def updateRoleSettings = testRunner.testCase.getTestStepByName("UpdateRoleSettings");
def deleteSession = testRunner.testCase.testSuite.project.getTestSuiteByName("common
GUI").getTestCaseByName("delete browser session");
////////////////////////////////////////////////////////////////////////////////////////////////////
for (int k = 0; k < functionalGroups.size(); k++)  {
    updateRoleSettings.setPropertyValue("assignedFunctionality", functionalGroups[k]);
    testRunner.runTestStepByName("UpdateRoleSettings");
    testRunner.runTestStepByName("Log in");

    if (functionalGroups[k].contains("Digitizing")) {
        testRunner.runTestStepByName("SelectAlternative");
        testRunner.runTestStepByName("EditSession[ON]");
    }
    testRunner.runTestStepByName("AddTab");
    testRunner.runTestStepByName("Click[Explorer]");
```

```
    click.setPropertyValue("IN-value", "//b[text()='features']");
    testRunner.runTestStepByName("Click");
    click.setPropertyValue("IN-value", "//span[text()='tee_route']");
    testRunner.runTestStepByName("Click");

    rightClick.setPropertyValue("IN-value", "(//tr//td[contains(@data-name, 'tee_route')])[1]");
    testRunner.runTestStepByName("RightClick");

    int j = 0;
    for (String b : contextButtons) {
        String allowed = contextButtonsAllowed[k][j++];
        checkElement.setPropertyValue("IN-isUp", "false");
        checkElement.setPropertyValue("IN-isAllowed", allowed);
        checkElement.setPropertyValue("IN-value", b);
        testRunner.runTestStepByName("CheckElement");
    }
////////////////////////////////////////////////////////////////////////////////////////////////////
    if (functionalGroups[k].contains("Digitizing")) {
        testRunner.runTestStepByName("EditSession[OFF]");
    }
    testRunner.runTestStepByName("Log out");
    deleteSession.run(new com.eviware.soapui.support.types.StringToObjectMap (), false);
}
```

Výpis 5: Groovy Script TC001[Explr]

Skript je o hodně zkrácený, protože v plné verzi by zbytečně zabíral mnoho stránek, je zde tedy uvedena odlehčená verze, která kontroluje kontextovou nabídku pouze pro jednu feature.

Na začátku do proměnné functionalGroups uložím názvy jednotlivých oprávnění, které lze roli v aplikaci Titanium přiřadit. Některá oprávnění nemůžou býti osamocena a musí být pospolu, např. PlanPosting nemůže být bez oprávnění Digitizing.

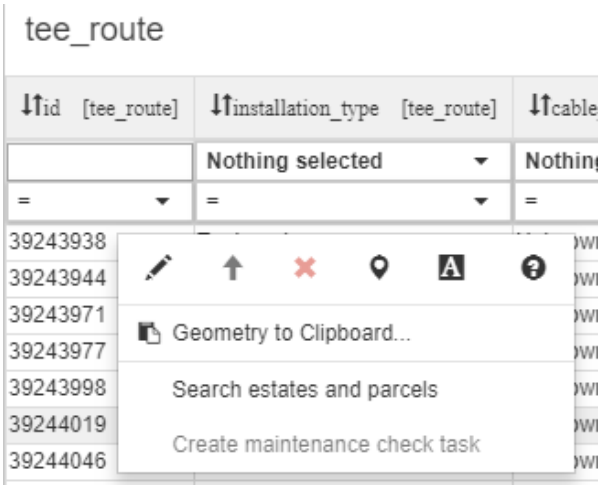
Poté proměnná contextButtons obsahuje řetězce jednotlivých xpaths, které identifikují jednotlivé položky kontextové nabídky, v tomto případě to budou tři položky.

Následující proměnné contextButtonsAllowedBase až po contextButtonsAllowedMaintenance obsahují poziční informaci o tom, která položka je povolena. Tyto proměnné si uložím do contextButtonsAllowed, který poté využiji v cyklu.

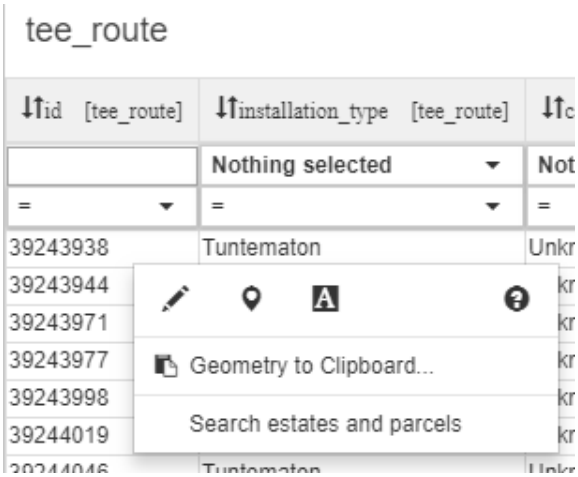
Další várka proměnných slouží k uložení instancí na jednotlivé kroky, které se budou ve zbytku skriptu používat.

Cyklus for bude iterovat od nuly až po počet různých oprávnění functionalGroups. Hned na začátku cyklu dojde k zavolání modulu UpdateRoleSettings, aby se nastavilo správné oprávnění.
Po nastavení oprávnění se uživatel přihlásí a pokud drží oprávnění Digitizing, načte alternativu a zapne EditSession. Důvodem je, že oprávnění Digitizing nemá smysl testovat bez EditSession, protože nedojde k zobrazení položek k němu příslušejících.

Otevře se Explorer a vybere se feature tee_route, na kterou se následně klikne pravým tlačítkem pro vyvolání kontextové nabídky.



Obrázek 38: Titanium – kontextová nabídka pro admina



Obrázek 37: Titanium – kontextová nabídka pro uživatele

Vnitřní cyklus for poté provede kontrolu jednotlivých položek. Všimněme si, že se nastavují properties IN-isUp a IN-isAllowed. Pak se spustí krok CheckElement, což je vlastně můj vytvořený modul check element IS or IS NOT allowed (business logic).

Pokud vše ve vnitřním cyklu for projde, opět se zkontroluje, zda nedrží uživatel oprávnění Digitizing, aby mohl skript EditSession vypnout.

Po podmínce dojde k odhlášení uživatele a smazaní instance prohlížeče, což je důležité, protože při každém novém přihlášení se vytváří nová instance prohlížeče, bez mazaní starých instancí by mohlo dojít k vyčerpání všech míst určených pro instance na node serveru.

Následně dojde k další iteraci cyklu for.

Obdobným způsobem jsem vytvářel ostatní testy testující bezpečnost kontextové nabídky, bohužel jsem některé testy nemohl vytvořit, protože k vyvolání kontextové nabídky vyžadují akce, které nejdou v SoapUI udělat.

7 Migrace testovacího nástroje na virtuální server

Migraci jsem opět podědil po svém předchůdci, nejednalo se o akutní záležitost, ale jelikož došlo k selhání node serveru, uvědomil jsem si, že mi může selhat i hub server. Pokud selže node server, je jeho nahrazení otázka chvilky, toto ale neplatí pro hub server, na kterém běží vlastní napsaný testovací nástroj NMA týmu a tj. celým názvem WMSTools – Universal Test Runner FrontEnd, zkráceně se na něj budu odkazovat pod jménem Test Runner.

Test Runner je aplikace napsaná v Ruby, jenž běží nad linuxovým serverem, aplikace se bohužel už dost dlouho nevyvíjí, takže je zastaralá, což přináší problémy při instalaci a konfiguraci na nové Linuxové OS.

Za úkol má zobrazovat výsledky testů a testy spouštět, buď na manuální vyžádání či automaticky dle nastaveného času.

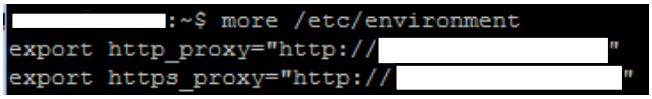
Takže jsem se rozhodl, že dokončím plán migrace na nový virtuální server, který na rozdíl od starého je pravidelně zálohován, takže při případném selhání je obnova serveru otázka pár kliknutí.

7.1 Konfigurace serveru

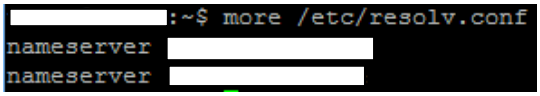
Ještě, než jsem se mohl pustit do instalace testovacího nástroje, musel jsem provést základní nastavení serveru, na kterém byl pouze nainstalován Debian verze 8.

Jako první věc jsem se tedy přihlásil na virtuální server a provedl nastavení proxy serverů a DNS serverů.

Proxy servery se v Linuxu nastavují jako systémové proměnné http_proxy a https_proxy, případně jiné dle použitého protokolu. Tyto proměnné jsem musel zapsat do souboru /etc/environment, který se načítá při každém přihlášení uživatele.



Obrázek 40: bash – vypsání environment



Obrázek 39: bash - vypsání resolv.conf

Co se týče DNS serveru, taky ty se zase zapisují do souboru /etc/resolv.conf

Po nastavení měl server přístup do Internetu.

Kromě těchto síťových nastavení jsem musel vytvořil nového uživatele, pod kterým poběží Test Runner. Na starém serveru vše běželo pod root účtem, což je velmi nebezpečné a špatné.

Nového uživatele jsem vytvořil pomocí příkazu ***useradd -m wmstools*** Byl jsem překvapen, když po přihlášení jsem neměl k dispozici žádný shell, při kontrole ***/etc/passwd*** jsem zjistil, že uživatel nemá nastavenou cestu k žádnému shellu.
Musel jsem tedy spustit ještě jeden příkaz a to ***chsh -s /bin/bash wmstools***

Kromě toho jsem chtěl, aby uživatel mohl používat sudo, bylo třeba balíček doinstalovat a provést nastavení, kde jsem skupině wheel povolil použití sudo a do skupiny jsem uživatele přidal.

```
# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
%wheel  ALL=(ALL:ALL) ALL
```

Obrázek 41: bash – nastavení sudo

7.2 Nastavení databáze

Kromě základní konfigurace serveru bylo nutné nainstalovat a nakonfigurovat databázi.
Test Runner jako databází používá MySQL, místo ní jsem se ale rozhodl nainstalovat MariaDB, která je následníkem MySQL a je zpětně kompatibilní.

Po instalaci jsem musel povolit v databázi použití innodb_file_per_table. Jedná se ve zkratce o to, že každá tabulka v databázi bude uložena jako jeden soubor, tím jsem si schopen zajistit kontrolu nad velikostí databáze. Pokud bude např. tabulka, ve které se drží výsledky testů příliš velká, tak ji jednoduše pomocí příkazu drop smažu a vytvořím znovu. Kdybych neměl innodb_file_per_table nastavený, tak by mi smazání nepomohlo, protože by se všechny tabulky ukládaly do jednoho souboru, který nelze zmenšovat. [14]

Nastavení se provede v souboru /etc/mysql/my.cnf kde se přidá ***innodb_file_per_table = 1***
Poté jsem musel databázi restartovat pomocí ***sudo /etc/init.d/mysql reload*** a kontrolu jsem provedl tím, že jsem se připojil do databáze a pustil příkaz ***show variables like 'innodb_file_per_table';***

Pak jsem mohl provést vytvoření databáze a oprávnění pro Test Runner:

```
systemctl start mariadb && systemctl enable mariadb
systemctl start mariadb.service
mysql -u root -p
create database wmstools;
create database wmstools_staging;
grant usage on *.* to wmsuser@localhost identified by 'SalvatorDali01';
grant usage on *.* to wmsuser@'%' identified by 'SalvatorDali01';
grant all privileges on wmstools.* to wmsuser@localhost;
grant all privileges on wmstools.* to wmsuser@'%';
grant all privileges on wmstools_staging.* to wmsuser@localhost;
grant all privileges on wmstools_staging.* to wmsuser@'%';
quit
```

Výpis 6: příkazy pro nastavení DB WMSTools

První dva příkazy zapínají MariaDB jako službu, tím pádem po restartu serveru dojde ke spuštění databáze. Následuje vytvoření databází wmstools a wmstools_staging a oprávnění na ně samotné.

7.3 Instalace RVM

Jak jsem již říkal, Test Runner je aplikace, která se už dlouho nevyvíjí, což znamená, že je závislá na staré verzi Ruby a starých verzí Ruby balíčků, které aplikace používá. Je to poměrně velký problém, protože Linux standardně funguje na balíčcích a jejich závislostech mezi nimi. Pokud bych se tedy snažil do Debianu nainstalovat starou verzi Ruby, musel bych do systému postupně přidávat jiné starší baličky, řešit konflikty mezi baličkami, až bych zjistil, že mam půlku sytému vyměněnou a druhou půlku systému nefunkční, a to nemluvím ani o tom, že některé starší balíčky ani už nemusí být běžně dostupné.

Do tohoto problému jsem se dostal a musel jsem jej vyřešit novou instalací systému a zkusit jinou cestu. Naštěstí jsem se dozvěděl o projektu RVM, který právě slouží k instalaci starých verzí Ruby, aniž by to mělo dopad na samotný systém.

https://rvm.io/

Instalace RVM včetně potřebné verze Ruby a jeho balíčků – gems.

```
command curl -sSL https://rvm.io/mpapis.asc | gpg2 --import -
\curl https://raw.githubusercontent.com/rvm/rvm/master/binscripts/rvm-installer | bash -s stable
        source /home/wmstools/.rvm/scripts/rvm
rvm install 2.0.0
rvm use 2.0.0
gem install bundler -v 1.13.7
gem install public_suffix -v 2.0.5
gem install do_mysql -v 0.10.17
gem install do_sqlite3 -v 0.10.17
gem install net-ssh -v 2.9.2
gem install puma -v 2.10.2
```

Výpis 7: příkazy pro instalaci RVM a nutných gems

Po instalaci RVM jsem se mohl pustit konečně k samotnému testovacímu nástroji.

7.4 Instalace testovacího nástroje Test Runner

Zdrojové soubory Test Runner se nacházejí na https://github.com/Kreny/wmstools/tree/NAM

Musel jsem vytvořit novou branche, kterou jsem si pojmenoval jako NAM_Virtual. Proč jsem vytvářel novou branche? Bylo to z důvodu, že kdybych upravoval přímo původní branche, tak bych si rozbil funkčnost testovacího nástroje na starém serveru, protože bylo nutné ve zdrojových souborech pozměnit IP adresu serveru, hostname, místo root uživatele nastavit wmstools uživatele a tak dále. Vše je zjistitelné z historie commitů.

Nová branch pro nový Test Runner: https://github.com/Kreny/wmstools/tree/NAM_Virtual

Po úpravě souborů jsem mohl přejít k naklonování projektu na server a provést instalaci.

Na server jsem musel doinstalovat balíček pro git, poté jsem vytvořil složku /home/wmstools/work/git/wmstools a spustil příkaz:

```
git clone -b NAM_Virtual https://github.com/Kreny/wmstools.git
```

Test Runner jsem měl tedy naklonovaný a mohl jsem spustit instalaci, která spočívala ve spuštění příkazu:

```
cd /home/wmstools/work/git/wmstools
bundle install
```

Výpis 8: příkazy pro kontrolu gems

Příkaz bundle install provede kontrolu, zda má aplikace přístup ke všem potřebným Ruby balíčkům.

Zjistil jsem například to, že mi chybí ještě balíček libmysqlclient-dev a tak jsem jej doinstaloval.

Poté jsem mohl spustit samotný deploy aplikace, tedy její nasazení a spuštění a to pomocí:

```
rake deploy_production_server
rake deploy_production_agents
```

Výpis 9: příkazy pro spuštění testovacího nástroje

Problém byl ovšem v tom, že mi selhal hned první příkaz, selhání bylo způsobeno tím, že aplikace nemohla spustit příkaz bundle, jako by jej neviděla.

Přijít na řešení nějakou dobu trvalo, protože při spuštění rake deploy se aplikace přihlásí přes SSH na server pod wmstools a provede své samotné spuštění. Simuloval jsem tedy tento proces,

kdy jsem se přihlásil pod účtem wmstools a spouštěl jednotlivé příkazy, které za rake deploy vězí, ale příkaz bundle mi procházel.

Nakonec jsem na to přišel, chyba byla způsobena kvůli environment variables. Když se pouští přes SSH příkaz, tak se použije neinteraktivní login a tím pádem se nenačte plno bash skriptů nastavující proměnné, a proto aplikace příkaz bundle neviděla. [15]

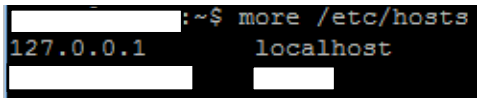
Nejjednodušší řešení je v sshd_config souboru nastavit **„PermitUserEnvironment yes“** a poté ještě spustit `env | grep rvm > ~/.ssh/environment`, který zařídí zkopírování všech RVM proměnných do SSH environment, který se použije při dalším neinteraktivním přihlášení přes SSH.

Po vyřešení problému s proměnnými mi první příkaz `rake deploy_production_server` prošel, ale druhý ne.

Řešení bylo oproti prvnímu problému mnohem jednodušší, zjistil jsem, že mi celý proces selhal na následujícím příkazu:

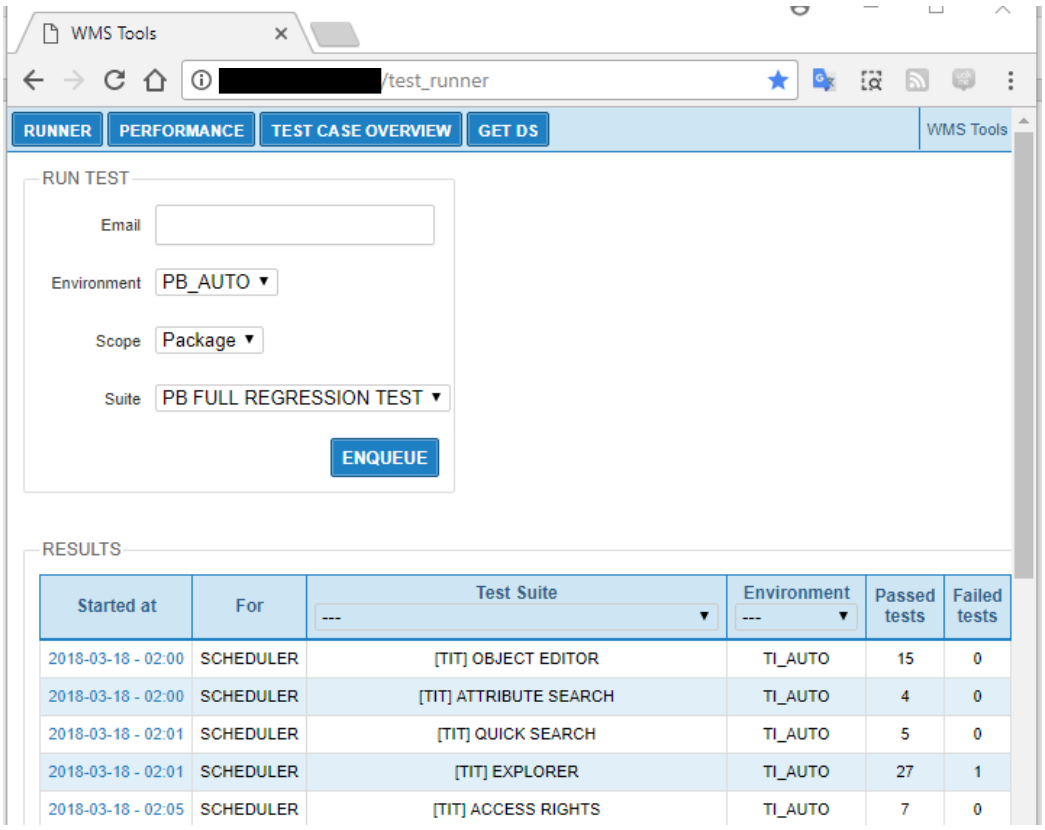
```
response = Net::HTTP.get(@hostname, '/rollback_running_executions', @port)
```

Aplikaci se nelíbilo hostname serveru a pak mi došlo to, že hostname serveru nemá nikde nastavený překlad na IP adresu. Přidal jsem tedy patřičný záznam do souboru /etc/hosts



Obrázek 42: bash – vypsání hosts souboru

A poté mi oba dva příkazy prošly a následně se Test Runner spustil:



Obrázek 43: WMS Tools

7.5 Konfigurace Selenium

Test Runner slouží pouze jako manager testů, sám žádné testy neparsuje ani nevykonává. Na vykonávaní testu slouží Selenium, které je nutné na server dostat.

Vytvořil jsem složku /home/wmstools/work/selenium a nakopíroval do ní soubor selenium-server-standalone-3.4.0.jar a hubconfig.json.

Samozřejmě jsem musel provést ještě instalaci Java prostředí.

Nakopírovaný jar soubor je samotný selenium server, který bude sloužit jako hub pro ostatní node servery a tím jim ukládat jaké testy má kdo vykonávat.

hubconfig.json obsahuje následnou konfiguraci [16]:

```
{
    "port": 4444,
    "newSessionWaitTimeout": -1,
    "servlets" : [],
    "withoutServlets": [],
    "custom": {},
    "capabilityMatcher": "org.openqa.grid.internal.utils.DefaultCapabilityMatcher",
    "throwOnCapabilityNotPresent": true,
    "cleanUpCycle": 5000,
    "role": "hub",
    "debug": false,
    "browserTimeout": 120,
    "timeout": 600
}
```

Výpis 10: hubconfig.json pro hub server

Pouze nastavení portu a role je to nejdůležitější.

Pak jsem musel vytvořit systemd službu, která by při případném restartu zajistila spuštění selenium hub serveru.

```
[Unit]
Description=Selenium Hub
After=syslog.target network.target


[Service]
Type=simple
User=wmstools
Group=wheel
ExecStart=/usr/bin/java -jar /home/wmstools/work/selenium/selenium-server-standalone-3.4.0.jar -role
hub -hubConfig /home/wmstools/work/selenium/hubconfig.json


[Install]
WantedBy=multi-user.target
```

Výpis 11: Systemd služba pro selenium

Soubor jsem pojmenoval jako seleniumHub.service a umístil jej do /etc/systemd/system

Následně jsem službu spustil pomocí příkazu ***systemctl start seleniumHub.service***

7.6 Instalace SoapUI

Měl jsem tedy již nainstalovaný Test Runner a nakonfigurovaný selenium hub server, chyběla mi však ještě poslední věc, a to parser.

Testy v SoapUI jsou uloženy ve formě XML souboru a tento soubor samozřejmě neumí Selenium číst, Test Runner pro parsování využívá backend linuxové verze SoapUI, který umožnuje být volán z různých aplikací.

Stáhl jsem tedy SoapUI aplikaci pro Linux z: https://www.soapui.org/downloads/latest-release.html
Použil jsem poslední verzi, tedy 5.4.0.

Dále jsem vytvořil složku /home/wmstools/work/soapui kde jsem provedl instalaci, která je jednoduchá, protože se jedná o .sh skript, který provede instalaci formou průvodce. Stačilo všude použít defaultní nastavení, jediná důležitá věc byla volba složky, do které se má aplikace nainstalovat.

7.7 Update SVN projektu

Sice jsem měl už vše nakonfigurované, narazil jsem ale na problém ohledně toho, že se mi nechtěl aktualizovat SVN projekt, ve kterém byly uloženy testy, před spuštěním testů.

Zjistil jsem, že je to z důvodu, že když Test Runner spustí příkaz na update SVN projektu, svn příkaz po něm chce jméno a heslo pro přístup do repository. Zkoušel jsem mnoho postupů, jak zajistit, aby si svn uložilo credentials a příště jej nevyžadovalo. Nakonec jsem ale upravil přímo příkaz, který provádí svn update na:

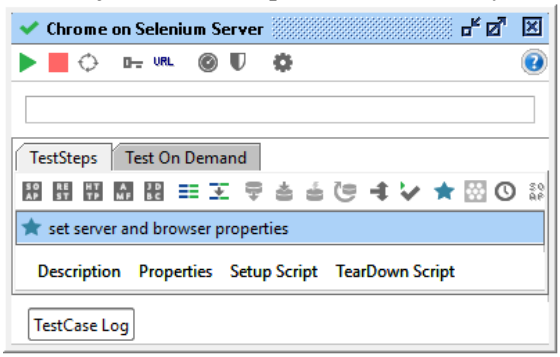
```
command = "svn update #{dir}/#{branch} --non-interactive --no-auth-cache --username uzivatel --
password heslo"
```

Tím jsem zajistil bezproblémovou aktualizaci SVN projektu.

Opět je vše zjistitelné v commitu na GitHub.

7.8 Úprava testů

Kromě konfigurace serveru jsem musel i upravit samotné testy, konkrétně šlo o modul:



Obrázek 44: TC Chrome on Selenium Server

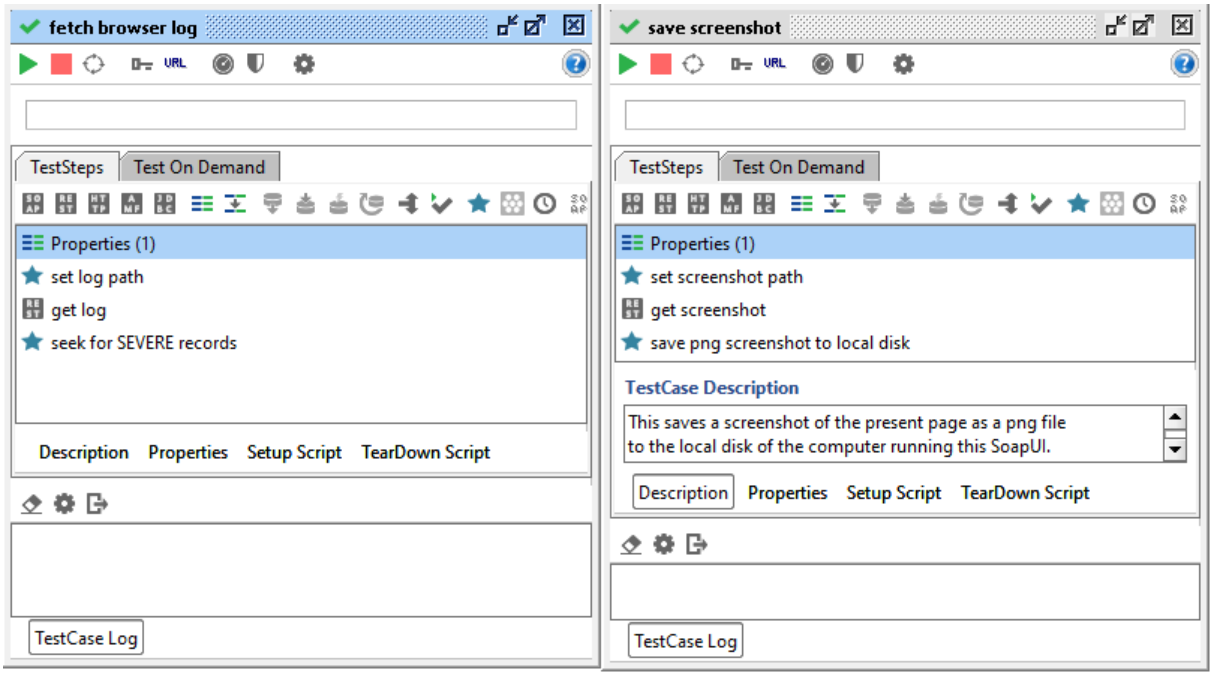
Kde jsem musel upravit Groovy skript:

```
testRunner.testCase.testSuite.project.setPropertyValue("SeleniumServerBase",
"http://IP_adresa_hubu:4444/wd/hub")
testRunner.testCase.testSuite.project.setPropertyValue("desiredCapabilities", '{"browserName":
"chrome","javascriptEnabled": true,"acceptSslCerts": true,"chromeOptions": {"args":["disable-
infobars"]}}')
```

Výpis 12: Groovy skript set server and browser properties

Bylo nutné nastavit nastavit IP adresu nového hub serveru.

A o moduly:



Obrázek 45: SoapUI – úprava modulů

U obou modulů bylo třeba upravit Groovy skripty tak, aby reflektovaly změnu uživatele, například:

```
def screenshotPath = context.expand( '${#Project#screenshotPath}' )
def path
def seleniumServerBase = context.expand( '${#Project#SeleniumServerBase}' )
if (!seleniumServerBase.contains('localhost') ) path =/* "/root/work/wmsTools/agent/"  +*/
screenshotPath
else path = screenshotPath
testRunner.testCase.getTestStepByName("Properties").setPropertyValue("screenshotPath",path)
```

Výpis 13: Groovy skript set log path I

na:

```
def screenshotPath = context.expand( '${#Project#screenshotPath}' )
def path
def seleniumServerBase = context.expand( '${#Project#SeleniumServerBase}' )
if (!seleniumServerBase.contains('localhost') ) path =/* "/home/wmstools/work/wmsTools/agent/"  +*/
screenshotPath
else path = screenshotPath
testRunner.testCase.getTestStepByName("Properties").setPropertyValue("screenshotPath",path)
```

Výpis 14: Groovy skript set log path II

Po úpravě testů jsem mohl otestovat nový Test Runner, takže jsem spustil pár testů a zjistil jsem, že mi Test Runner funguje.

Na rozjetí nového testovacího nástroje Test Runner jsem celkem hrdý, protože k němu neexistuje takřka žádná dokumentace a na mnoho věcí jsem musel přijít sám, případně metodou pokus/omyl.

Díky novému testovacími nástroji jsem zkvalitnil testovací infrastrukturu.

Závěr

Využité dovednosti získané v průběhu studia
Dovednosti, které jsem zejména využil byly hlavně čerpány z předmětu *Správa operačních systému*, které jsem využíval při snaze o zprovoznění nového testovacího nástroje a *Úvod do databázových systému*, kde se mi pokročilá znalost SQL často hodila při psaní GUI testů.

Chybějící znalosti
Zejména testovaní jako takové a práce s TFS – systém pro správu softwarových projektů.
Také mi chyběla znalost SoapUI.

Dosažené výsledky v průběhu praxe a zhodnocení
Jsem rád, že jsem se rozhodl absolvovat praktickou bakalářskou práci. Nejen, že jsem si rozšířil znalosti ohledně automatizovaného testování, zejména GUI, ale také jsem velmi významným dílem přispěl k jejich udržení a přežití.

Zvládl jsem provést úspěšnou migraci testů a špatně napsané testy opravit, tím jsem jim o mnoho zvýšil jejich kvalitu a pokud někdy v budoucnu dojde například ke změně podkladových dat v databázi, případné opravy už nebudou tak bolestné.

Dále jsem vymyslel způsob, jak detekovat, že nedošlo k nalezení odpovídajícího elementu, aniž by test selhal, což jsem využil při tvorbě testů testujících kontextovou nabídku.

Dokázal jsem zprovoznit nový Test Runner a tím zajistit GUI testům stabilní zálohované a správně nakonfigurované prostředí.

A nakonec jsem poznal, jak funguje vývoj velkého softwaru a jak testeři spolu s programátory spolupracují. Byl to pro mě naprosto odlišný styl práce, než na který jsem byl zvyklý z předchozí pracovní pozice, kde jsem pracoval jako Windows administrátor.

Praktická bakalářská práce ve mně zanechala pocit z dobře odvedené práce a byla pro mě dokonalou přípravou na to, co bych v IT do budoucna chtěl dělat a na co se zaměřit.

Literatura

[1] O nás. *Tieto - IT, výzkum a vývoj a poradenství.* [Online] Tieto Czech s.r.o., 2016. https://www.tieto.cz/tieto-o-nas.

[2] Selenium (software). *Wikipedia, the free encyclopedia.* [Online] Wikimedia Foundation. https://en.wikipedia.org/wiki/Selenium_(software).

[3] Stewart, Simon a Burns, David. Protocol. *WebDriver.* [Online] W3C, 30. Březen 2017. https://www.w3.org/TR/webdriver/#protocol.

[4] What is SoapUI? *The World's Most Popular API Testing Tool | SoapUI.* [Online] SmartBear Software, 2018. https://www.soapui.org/open-source.html.

[5] Scripting and the Script Library. *The World's Most Popular API Testing Tool | SoapUI.* [Online] SmartBear Software, 2018. https://www.soapui.org/scripting-properties/scripting-and-the-script-library.html.

[6] Robie, Jonathan, Dyck, Michael a Spiegel, Josh. Introduction. *XML Path Language (XPath) 3.1.* [Online] W3C, 21. Březen 2017. https://www.w3.org/TR/xpath-31/#id-introduction.

[7] Locating UI Elements (WebElements). *Selenium WebDriver — Selenium Documentation.* [Online] Selenium Project, 20. Březen 2018. https://www.seleniumhq.org/docs/03_webdriver.jsp#by-xpath.

[8] Introduction. *The Apache Groovy programming language - Groovy reference documentation.* [Online] Apache Groovy project, 23. Březen 2018. http://groovy-lang.org/single-page-documentation.html.

[9] What is ArcMap? *ArcMap | ArcGIS Desktop.* [Online] Environmental Systems Research Institute, Inc., 2016. http://desktop.arcgis.com/en/arcmap/10.3/main/map/what-is-arcmap-.htm.

[10] Dynamic map service layer. *ArcGIS for Developers.* [Online] Environmental Systems Research Institute, Inc., 2017. https://developers.arcgis.com/macos/10-2/objective-c/guide/arcgisdynamicmaplayer.htm.

[11] Environment handling in soapUI. *The World's Most Popular API Testing Tool | SoapUI.* [Online] SmartBear Software, 2018. https://www.soapui.org/soapui-projects/environment-handling.html.

[12] Robie, Jonathan, Dyck, Michael a Spiegel, Josh. Filter Expressions. *XML Path Language (XPath) 3.1.* [Online] W3C, 21. Březen 2017. https://www.w3.org/TR/xpath-31/#id-filter-expression.

[13] Lindstrom, Jan. MariaDB 10.1.1: Defragmenting unused space on InnoDB tablespace. *MariaDB.org - Supporting continuity and open collaboration.* [Online] MariaDB Foundation, 24. Říjen 2014. https://mariadb.org/defragmenting-unused-space-on-innodb-tablespace/.

[14] BARRETT, Daniel J., SILVERMAN, Richard E. a BYRNES, Robert G. *SSH, The Secure Shell: The Definitive Guide: The Definitive Guide.* místo neznámé : O'Reilly Media, Inc., 2005.

[15] Selenium Grid. *Selenium Documentation.* [Online] Software Freedom Conservancy, 18. Leden 2018. https://seleniumhq.github.io/docs/grid.html.